Decision Support for Planning of Multimodal Transportation with Multiple Objectives

Petersen, Hanne Løhmann

Publication date:
2009

Decision Support for Planning of Multimodal Transportation with Multiple Objectives

Hanne L. Petersen

Kongens Lyngby 2009

PHD-2009-03
Department of Transport
Technical University of Denmark

Published by: DTU Transport
Bygningstorvet 116 Vest
DK-2800 Kongens Lyngby
Denmark

Order at: http://www.transport.dtu.dk
Tel +45 45256500, Fax +45 45936412
transport@transport.dtu.dk

ISSN: 1601-9458 (Electronic version)
ISBN: 978-87-7327-196-4 (Electronic version)

ISSN: 1600-9592 (Printed version)
ISBN: 978-87-7327-197-1 (Printed version)

Summary

This thesis treats two different planning problems from the transportation industry; one from freight transport and one from passenger transport. Each problem emerges as a combination of problems that are already known from the operational research literature, and introduces a new view of well-known issues. They both originate in the world of multimodality, and deal with problems that arise as a consequence of the combined use of several modes.

The thesis introduces the Double Travelling Salesman Problem with Multiple Stacks (DTSPMS), which is a problem that combines routing and last-in-first-out loading constraints. After giving an introduction to the problem, a range of related problems from the literature are discussed. Some considerations are made regarding basic bounds for the problem, and illustrations of problem solutions are given to provide an impression of how solutions of the DTSPMS compare to solutions of the regular Travelling Salesman Problem. Next, two papers are presented, introducing respectively heuristic and exact solution procedures for the problem. The heuristic approach tests a variety of metaheuristic solution approaches, of which a large neighbourhood search obtains the best results. Results are provided for real-life instance sizes, for smaller instances for which the optimal solution value is known, and for some larger instances, which can also be justified from a real-life perspective. With the purpose of solving the DTSPMS to optimality, several different mathematical formulations are presented and tested in the second paper. The most promising approach is based on a decomposition of the problem into a routing part and a loading feasibility part, and all tested instances with 15 orders can be solved using this approach.

The Simultaneous Vehicle Scheduling and Passenger Service Problem (SVSPSP) is an integration of two problems that are usually solved separately and sequentially, namely the timetabling problem and the Vehicle Scheduling Problem. The SVSPSP allows for the solution of the timetabling problem to be reoptimised when considering the vehicle scheduling phase, and considers passenger inconvenience at transfers at the same time. The paper presents a mathematical model of the prob-

lem, and the implementation of a large neighbourhood search solution procedure. The problem is solved for a real-life based problem instance, containing eight bus lines in the Greater Copenhagen area, and the results are promising.

Resumé

Nærværende afhandling, "Beslutningsstøtte til planlægning af multimodal transport med flere målsætninger", behandler to forskellige planlægningsproblemer fra transportsektoren; et fra godstransport og et fra passagertransport. Begge problemer er opstået som en kombination af problemer der allerede er velkendte fra operationsanalyselitteraturen, og bidrager med en ny synsvinkel på velkendte problemstillinger. De er begge af multimodal natur, og beskæftiger sig med problemer der opstår som en konsekvens af den kombinerede anvendelse af flere transportformer.

Afhandlingen introducerer det dobbelte handelsrejsendes problem med flere stakke (double travelling salesman problem with multiple stacks – DTSPMS), som kombinerer rutelægning og LIFO-pakningsbetingelser (last-in-first-out). Efter en introduktion af problemet, diskuteres en række relaterede problemer fra litteraturen. Der præsenteres nogle overvejelser vedrørende øvre og nedre grænser for løsningsværdier, og illustrationer af nogle løsninger præsenteres for at give en fornemmelse af forholdet mellem løsninger til DTSPMS og til det traditionelle handelsrejsendes problem. Herefter præsenteres to artikler, der introducerer henholdsvis heuristiske og eksakte løsningsmetoder til problemet. Den heuristiske tilgang undersøger flere forskellige metaheuristiske løsningsmetoder, hvoraf en stornabolagssøgning (large neighbourhood search) opnår de bedste resultater. Der præsenteres resultater for probleminstanser af realistisk størrelse fra den virkelige verden, for mindre instanser, hvor den optimale løsning er kendt, og for større instanser, hvis størrelse ligeledes kan retfærdiggøres ud fra et virkelighedsnært perspektiv. Med henblik på at opnå optimale løsninger til DTSPMS introduceres og afprøves forskellige matematiske formuleringer i den anden artikel. Den mest lovende løsningstilgang er baseret på en dekomponering af problemet i en rutelægningsdel og en pakningsdel, og alle testede probleminstanser kan løses med denne metode.

Det kombinerede vognløbsplanlægnings- og passagerserviceproblem (simultaneous vehicle scheduling and passenger service problem – SVSPSP) integrerer to problemstillinger som sædvanligvis betragtes adskilt og

sekventielt, nemlig et køreplansproblem og et vognløbsproblem. SVSPSP åbner mulighed for at køreplansproblemet kan reoptimeres i forbindelse med vognløbsplanlægningsfasen, og betragter samtidig passagergener ved skift. Artiklen præsenterer en matematisk model for problemet, og en stornabolagsløsningsmetode implementeres. Problemet løses for en virkelighedsbaseret probleminstans som indeholder otte S-buslinier samt S-togene fra københavnsområdet, og resultaterne er lovende.

Preface

This thesis was prepared at the Technical University of Denmark in partial fulfillment of the requirements for acquiring the Ph.D. degree in engineering. The work was carried out at the 3 departments DTU Transport (previously Centre for Traffic and Transport), the former Informatics and Mathematical Modelling, and DTU Management Engineering, with the two latter as official place of employment (before and after January 1, 2008).

The thesis introduces two new research problems within the area of operational research in transportation planning, and considers exact and heuristic solution methods to solve each of them.

The thesis contains three research papers, of which two have been accepted for publication, and one has been conditionally accepted for publication. In addition to the three papers there is a supplementary report, which contains further background and details for the papers and the research topics covered therein.

Acknowledgements

I wish to thank my main supervisor Professor Oli B.G. Madsen for his guidance and support during my work on this project, and for his faith in my ability to finish it. I would also like to thank my co-supervisor Professor Jens Clausen for giving valuable input whenever I asked for it.

The external stay for this Ph.D. was spent at the Università degli Studi di Brescia, and I wish to thank Professor Grazia Speranza and Assistant Professor Claudia Archetti for being a fantastic inspiration for my work during my time in Brescia. I would at the same time like to thank the rest of the group for making my visit a pleasant experience.

I would also like to thank The NABIIT Programme Commission (Nano Science and Technology, Biotechnology and IT) from The Danish Council for Strategic Research for funding the project.

Furthermore, I would like to thank Finn Laursen from Easy Cargo Systems A/S, for introducing me to the problem that was later dubbed the double travelling salesman problem with multiple stacks, and for providing a lot of inspiration and industry background knowledge at the early stages of the project.

During the last years I have been surrounded by various colleagues at CTT, IMM, MAN and DTU Transport – luckily most of my colleagues have changed less often than the names of their departments. I want to thank them all for their encouragement and support throughout the project, and to the ones I have worked more closely with, a special thanks is due for their inspiration and discussions on all sorts of topics.

Kgs. Lyngby, June 2009

Hanne L. Petersen

Papers included in the thesis

[A] Hanne L. Petersen, Oli B.G. Madsen. The Double Travelling Salesman Problem with Multiple Stacks - Formulation and Heuristic Solution Approaches. *European Journal of Operational Research*, 2009. Accepted for publication.

[B] Hanne L. Petersen, Claudia Archetti, M. Grazia Speranza. Exact Solutions to the Double Travelling Salesman Problem with Multiple Stacks. *Networks*. Accepted for publication.

[C] Hanne L. Petersen, Allan Larsen, Oli B.G. Madsen, Bjørn Petersen, Stefan Røpke. The Simultaneous Vehicle Scheduling and Passenger Service Problem Conditionally accepted for publication.

Contents

Chapter 1

Introduction

Transportation of goods and passengers plays an important role in the society of today, and with increased globalisation the dependency on transport is not likely to reduce in the coming years. The transportation industry currently accounts for 7% of the GNP of the European Union, and 30% of the energy comsumption.[1] Total freight transportation within the European Union amounted to almost $4 \cdot 10^{12}$ tonne-kilometres in 2005, showing a 30% increase in the period 1995–2005. 44% of this was road transport, 42% waterborne, and 8.5% rail.[2]

Recent years have shown an increased awareness of environmental issues, which call for a reduction in the ressource consumption used for transportation. This can be achieved either by reducing the use of transport (amount or distance), or by more efficient ways of providing the desired level of transportation (be it planning or technical improvements). Furthermore, road congestion is a growing problem world-wide, and this problem cannot be solved exclusively by expanding infrastructure; there is also a need to reduce traffic on the roads. At the intra-continental level both of these goals can be obtained by moving goods and passengers off the roads, and onto other modes, such as rail, or, where applicable, short-sea/inland shipping. For passenger transportation this typically implies increased use of public transportation to replace individual car travel; even though buses travel on the same roads as cars, they cause much less congestion per passenger. For freight transport, it is a declared goal of the European Commission to increase usage of the rail and sea modes.[3]

[1] Transport: In Brief, http://www.euractiv.com/en/transport/transport-brief/article-159326, retrieved Feb 2, 2009

[2] Panorama of Transport, http://epp.eurostat.ec.europa.eu/cache/ITY_OFFPUB/KS-DA-07-001/EN/KS-DA-07-001-EN.PDF, retrieved Feb 2, 2009

[3] "EU seeks to shift freight to rail and shipping",

However, since these modes are usually unable to provide door-to-door transportation, they can only be used in combination with road transport. For such a combination to be a viable alternative, the transfer of goods between modes has to be performed smoothly and without significant increases of cost or time.

Intermodal transportation is the term used to describe any transportation process which requires the use of more than one mode, and is thus dependent on the possibility of performing modal transfers. These modal transfers, and the terminals at which they take place, play an important role in intermodal transportation, since they are a necessity, a potential bottleneck, and a key feature of the intermodal transportation chain.

Intermodal freight transportation has been in use for a number of years, but has received increased attention recently, due to escalating congestion and environmental considerations. The ability to handle intermodal transfers is closely related to the availability of uniform storage equipment, that enables terminals to handle many different types of goods. With the use of uniform containers, each terminal only needs the ability to handle the containing unit, regardless of its contents, which significantly reduces the requirements for the handling equipment. Such standardised containers have been developed during the 20th century, and are in widespread use today.

In passenger transportation, the combined use of several modes for one journey is also no novelty, and use of public transportation has always required combination with other modes (walking to the nearest embarkment point, if nothing else). With the complex networks of public transportation that are available in many modern cities, the usefulness of intermodal travelling has increased, and today it is necessary to consider several modes (train, bus, tram, metro, etc.) for almost any form of local or regional public transport.

This thesis will consider two different problems that both originate in an intermodal setting; one from freight transportation, and one from public transport. The first is a problem that is related to the use of intermodal freight transportation, and occurs as a part of an intermodal chain, where the initial and final phases are concerned with the consolidation and breaking up of containerised transport. The other problem originates in multimodal passenger transportation, and considers the use of scheduling to help reduce passenger inconvenience at modal transfers.

1.1 Multimodality

In order to reduce the amount of traffic on the roads, a part of this traffic must be transferred to other modes. However, since trucks are

http://www.euractiv.com/en/transport/eu-seeks-shift-freight-rail-shipping/article-167734, retrieved Feb 2, 2009

often the only mode capable of performing the end parts of a freight transport, i.e. the actual visit at the customer site, the use of alternative modes can often only happen in combination with trucks. Reducing the amount of traffic in passenger transportation implies an increased use of public transport, and multimodality is a cornerstone in a flexible public transport network. Thus, multimodality becomes an important factor in removing traffic from the roads – concerning freight as well as passenger transport.

Public transportation depends heavily on the availability of good and plentiful opportunities for transfer between the different bus and train lines that are available. A good network for public transportation is often built around a rail backbone, which can transport large numbers of passengers much more easily than a bus network can, while buses are typically used when transporting passengers between a train station and their home or working place. Thus the interchange between trains and buses is often inevitable in a well-functioning public transportation network, and short interchange times are consequently a very important factor in making public transportation a viable alternative for personal travel.

In freight transportation trucks serve the same purpose as buses do in public transport – they can access almost any location to pick up or deliver orders, but have some disadvantages for transportation of large quantities over longer distances. In freight transportation there is often a considerable potential economical benefit obtainable from economies of scale when using rail or sea for long-haul transportation, albeit often at a cost of reduced flexibility and speed. Furthermore, the final destination may be practically unreachable by use of truck alone; this is naturally the case for many intercontinental shipments, but also around e.g. the Mediterranean or the Baltic Sea.

Intermodality has the potential to be successfully applied in many transportation contexts, but naturally it also has its limitations. The transfers and consolidation occuring in intermodality inevitably take time, meaning that intermodality is usually not competitive for highly time-critical or urgent deliveries. This naturally disregards transports where an intermodal combination does indeed provide the fastest opportunity, such as air transport of freight that is combined with trucking to and from the airport.

Intermodality is, or can be, a viable alternative for a large amount of the freight transportation occuring today, where timeliness is a factor, but not the most important factor. This includes inter-continental containerised transports which are by their very nature intermodal, since no single mode is available that can perform the entire transport. Also for door-to-door parcel delivery, intermodality and consolidation play an important role. In most cases speed is an issue for such tasks, albeit not at any cost. This sets the stage for parcel carriers, who often operate an elaborate net of terminals in a hub-and-spoke network. In such networks

parcels are often consolidated at several levels, and at each terminal the consolidated units may be split up or combined to form new units. This means that the level of consolidation varies during each leg of the journey. In such a network all legs may be covered by trucks, however due to the re-loading and consolidation of the goods, it still exhibits many of the properties of an intermodal transportation chain.

In passenger transport the temporal aspect is generally more important than when handling freight. A travelling passenger will usually not accept an excessively long waiting time for a transfer, whereas a container does not experience the actual travel, and its journey will usually be acceptable as long as the departure and arrival times (and thereby trip duration) are acceptable. For passengers, the value of time is much higher than for freight, and intermodality will only be considered if it provides the fastest or otherwise "best" journey. However, since the transfer operation is much less demanding in passenger transport, the bottleneck effect is smaller for passengers than for freight, and multimodality can often be a competitive alternative in terms of journey duration, e.g. in commuting using public transportation.

1.1.1 Freight transport

A key feature of intermodal freight transportation is the use of consolidation to obtain economies of scale. By consolidating, it becomes possible and economical for the transporter to use modes such as rail or sea, where the cost of transportation per unit is lower, but where the capacity and start-up costs are such that most customers depend on using it as a service shared with others. For consolidation to become a popular and widespread option, it is necessary to have standardised packaging equipment, since the use of such equipment immensely facilitates the required handling operations. Such packaging equipment is used today, in the form of standardised containers. The most widely used standard container size used is 40 ft (approximately 12 m), which gives a container with a loading capacity of 2 TEU (twenty-foot equivalent units[4]), allowing approximately 34 m^3 and up to 30 tonnes. Such 40 ft containers, along with the 20 ft variant (1 TEU), enclose the vast majority of international containerised goods today. Some variations of the standard 40 ft container exist, including extra long (45 ft, around 14 m) and extra high containers.

Like containers, pallets provide a standardised way of packaging freight, facilitating handling and planning. When goods are packaged on a pallet and securely wrapped, it constitutes a loading unit which takes up a fixed amount of floor area, making planning easier than when irregular items are handled. Furthermore the use of pallets ensures that the goods can be easily handled and moved by e.g. a forklift truck. The standardised

[4] A *TEU* is the amount of goods that can be transported in a 20 ft container, and is the standard measure for amounts of transported cargo

europallet (not a worldwide standard like the 40 ft container) measures 80 by 120 centimetres.

In freight transportation, cargo is often categorised as truckload, less-than-truckload, bulk or break bulk. *Truckload* (or *containerload*) refers to orders where the customer has enough cargo to fill a truck with one order. The truck can then travel directly from the origin to the destination, without any need for consolidation. When the orders are smaller they are referred to as *less-than-truckload* (LTL). An LTL order often consists of a number of pallets to be transported, which means that all items handled by the shipper are still of identical shape and size, though other objects of more irregular shape are also classified as LTL. Finally, *bulk* refers to such goods as grain or coal, and *break bulk* to items that can be handled individually, such as boxes or barrels. Pallets are also usually considered break bulk.

When considering freight transportation, there are typically four modes available to the planner: truck, rail, sea and air. Of these alternatives truck is by far the most flexible, since all other modes require designated terminals (rail terminal, harbour or airport) for goods to be loaded and unloaded. However the cost of this flexibility is high, for example in terms of man-power, since a truck can in general only transport 2 TEU[5] at a time. For comparison a freight train can typically transport around 80–100 TEU, or twice that amount when using double-stacking[6], and the largest container ships have a capacity over 10,000 TEU, with a crew of 10–20 persons. For freight airplanes the capacity lies somewhere between that of trucks and trains, however the price of transportation by air is much higher than for any other mode. The sea and air modes may be necessary to reach certain locations that are inaccessible by road or rail alone (such as intercontinental connections), though typically truck transport is still required for parts of such a journey. Air transport has the additional advantage of being high speed, although the speed comes at a high monetary cost. As a result, air transport is not an alternative that is considered for much of the large quantity of goods that is being transported around the world.

The issue of intermodality in freight transportation is typically relevant for all but the very shortest deliveries. The use of trucks for customer visits is often inevitable, since the customer is rarely located directly at a suitable terminal. However for further transport of containerload freight, rail or sea, depending on the final destination, is often a good alternative for economical reasons.

[5] in some places in Europe up to 3 TEU are allowed, and in other parts of the world even more. However the larger capacities are only applicable for long-haul transportation, and it seens fair to assume a maximum capacity of 2 TEU for trucks that perform customer visits.

[6] double-stacking allows the placing of containers on top of each other, thus doubling the capacity of a train of a given length. The use of double-stacking in Europe is very limited due to the number of low bridges etc., but it is in more widespread use in other parts of the world.

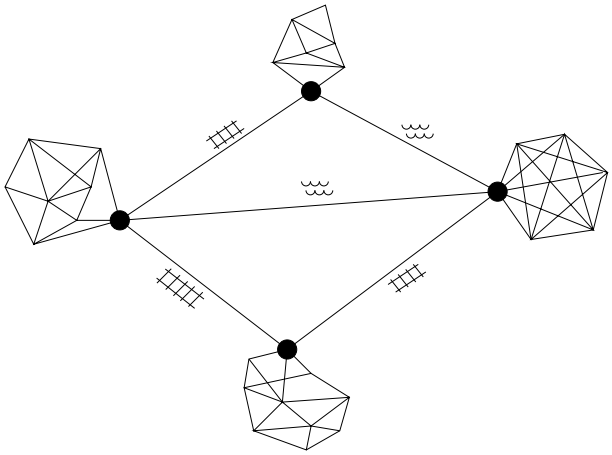


Figure 1.1: An example of a small intermodal freight network.

Figure 1.1 shows a sketch of an sample intermodal network. It consists of a number of separated distribution networks, each served by trucks, and each with its own terminal. The terminals are linked together by rail or ship routes.

Intermodal freight transportation presents some requirements for the terminals that handle the mode transfers, where handling equipment and cranes must be available to perform the transfer operations. These requirements impose a cost on intermodal transportation, meaning that a certain volume of goods must be present for intermodality to become a viable alternative. Furthermore the planning issues at and around such a terminal introduce a wide range of non-trivial planning problems, that lie outside the scope of this thesis.

An introduction to intermodal freight transportation, along with a review of the relevant problems and modelling approaches from the operations research literature can be found in Crainic and Kim [24]. Other reviews on the use of operations research in intermodal freight transportation are available in Macharis and Bontekoning [77], Bontekoning et al. [7], and Caris et al. [10].

1.1.2 Passenger transport

In personal transport the available modes can roughly be divided into foot, bicycle, car, bus, rail, sea and air. Among these modes sea and air are typically only used for occasional, long-distance transportation, and then generally in combination with one or more of the other modes.[7] Bus and train usually constitute the public, local transport, while travel by foot, bicycle (and motorised variants thereof), and car provide individual means of transport.

[7] In certain areas short-distance ferry services are also part of everyday commuting. Air commuting exists but is less common by far than commuting by bus and train.

The use of public transport for a journey usually also requires a (small) amount of individual transport, typically to get to and from the nearest bus stop. This makes any use of public transport multimodal by nature; the individual transport e.g. between the home and the first terminal/embarking point of a public transport vehicle must necessarily employ some other mode. This journey can often be performed by foot, but may also be considered an integral part of the whole transportation chain. For example a traveller may choose not to use the nearest station, but instead use one that possesses other qualities, such as better parking or shopping opportunities. However such considerations lie outside the scope of this thesis, which will only consider a problem where the passenger embark and disembark points have been determined beforehand, and are only mentioned to illustrate the complexity of the multimodal transportation chain.

In most urban areas, public transportation is used by a large number of commuters in everyday life, and a lot of passengers are affected by its daily operation. It is often necessary for passengers to perform transfers to reach their final destination when travelling, and hence the effects of multimodal public transportation are noticeable to many. This implies that there are considerable gains to be made if the quality of interchange in public transportation can be improved.

In passenger transportation the mode change is less equipment intensive compared to freight transport, since passengers generally transport themselves between the modes, which means that one of the major obstacles of intermodal freight transportation is not present. Instead passengers experience a certain inconvenience by having to change vehicle – an inconvenience which can be reduced considerably if the arrival and departure times are well-matched to reduce waiting time, and if the physical layout of the terminals is suitably structured. Thus the constraints and objectives considered in multimodal public transportation are not identical to those considered in intermodal freight transportation. Terminal layout is an example of a planning problem that occurs within both passenger and freight multimodal transportation, but with different constraints. Freight terminals need space for handling equipment and (temporary) storage, whereas passenger terminals may focus more on short walking distances, intuitive paths and easily accessible waiting rooms.

1.1.3 Intermodality vs. multimodality

In freight transportation the terms intermodality and multimodality can often be used interchangeably, however some differences of nuance exist.

The term multimodality (*multi*, from Latin, *many*) can be used to refer to any situation where more than one mode is available. It does not always require that multiple modes are used, and can also refer to situations where different tasks use different modes.

Intermodality (Latin: *inter*, roughly *between*) refers to a situation where interaction and changes between modes occur, and thus places more focus on the transfer situation.

The United Nations[8] uses a definition of multimodality as "carriage of goods by two or more modes of transport" and intermodality as "the movement of goods in one and the same loading unit or road vehicle, which uses successively two or more modes of transport without handling the goods themselves in changing modes", thus stressing the (non)-handling of the goods in the transfer situation. Similar definitions are used by the UN, EU and OECD. The main distinction between these definitions seems to be that intermodality is required to use some form of uniform packaging (e.g. containers) of the goods which facilitates transfers, by enabling transfer without direct handling, while multimodality is not concerned with the means of obtaining the modal combination. Thus transportation of bulk materials, such as grain or coal, and liquid materials would be an example that can be classified as multimodal, but not intermodal, transportation using the above-mentioned definitions.

Crainic and Kim [24] refer to a similar definition, and point out that it is too restrictive, by ruling out mail services which include sorting operations at terminals. However, an alternative interpretation would be that since the sorting is not performed as part of the mode change itself, this situation is not ruled out by the UN definition. As long as the mail is grouped (as part of the sorting) this does indeed provide ways of handling the freight in larger quantities, without handling each item individually. The alternative definition suggested by Crainic and Kim refers to "the transportation of less-than-vehicle-capacity loads by nondedicated services [using multiple modes], as well as transfer activities between these modes in dedicated terminals", thus focusing on consolidation and on terminal activities.

In passenger transport only the term multimodality appears to be commonly used, and since the issues of packaging and transfer handling are irrelevant, a distinction seems meaningless.

In the context of this thesis the terms intermodality and multimodality will be used interchangeably, since the mode choice is not considered per se, and is irrelevant for the purposes considered here. Instead the modes and transfers to be used have already been determined, and the models and solutions that are considered are based on these pre-determined modal decisions. Thus the main parts of the thesis will deal with intermodal problems, with the focus being on the side-effects that occur as a consequence of the use of intermodality, rather than the intermodality itself.

[8] UNECE, Terminology on Combined Transport, Economic Commission for Europe, 2001

1.2 Problem objectives

When solving optimisation problems, the choice of a suitable objective function is often not obvious. Several measurements for the quality of a given solution may exist – in transportation this is often a matter of cost versus duration and/or customer inconvenience, where the customers in question can be either passengers or senders/receivers of freight.

Often such problems can be reduced to a single significant objective function, either because the problem owner is not interested in other aspects, or because one objective is much more prominent than the others. Once a single objective has been determined, any given solution can be evaluated with respect to this objective, and compared to other solutions, and the search for an optimal or good solution can begin.

When problems cannot be reduced to a single objective, one must turn to methods that are capable of handling multiple solution objectives simultaneously. Multiple objectives can either be tackled by a preprocessing procedure which constructs a single representative objective (typically involving conversion between different measures), or by use of a multicriteria approach, which returns a range of solutions, and leaves the final decision to external (typically human) factors and qualitative judgement.

1.2.1 Multiobjectivity

In many situations there exist several potential objectives that can be evaluated when assessing the value of a solution, and the decision about how to select one or more of these is not trivial.

Single-objective, deterministic problems with a well-defined objective can lead rather directly to the formulation of a model, which can then (ideally) be solved to produce a single optimal solution. However, when several candidate objectives are present, one must first determine how these should be handled, and the solution process will typically consist of at least two phases.

Several possibilities exist:

1. Constructing a model that contains all objectives, and determine a set of so-called *Pareto*-optimal solutions.
2. Convert all objectives into one common measure or scale, such as money or utility.
3. Transform one or more objectives to constraints by imposing a bound, and solve with regard to the remaining objectives.
4. Appoint one among the existing objectives as the most significant, and construct the model to solve the problem with regard to this main objective. The secondary objectives can then be considered as part of a post-processing phase.

The first approach is a “true” multiobjective approach (see e.g. Ehrgott

and Gandibleux [32] for a survey up to 2000), while the three other suggestions provide different ways of solving a problem using a traditional single-objective solution method. Each of these approaches has its own advantages, and which approach one chooses in a particular case should depend on the problem at hand, and the situation in which it arises. Methods 1 and 4 lead up to a post-processing phase where several potential solutions may be presented to the decision maker, whereas methods 2 and 3 will attempt to handle all objectives simultaneously, and only return one solution.

The choice of how to deal with several potential objectives is an important decision for any problem solving process, and should be made in cooperation with the problem owner(s).

1.2.2 Selecting an objective

In many real-life problems there are several parameters which express the quality of a solution for different stakeholders, and thus these problems are multiobjective if viewed from a global perspective. However, often the decision maker will mainly be interested in one objective, and the remaining stakeholders will not be able to influence the final decision.

In cases where the decision maker is a public institution wishing to have local transportation needs covered, the overall objective can often be divided into primary objectives (cost) and secondary objectives (level of service, passenger effects); from an operator point of view the target is to operate an acceptable schedule at the lowest possible cost, not to determine the schedule that provides the best possible passenger service. A certain minimum level of service is often included in the contract with the service provider, and the problem is to cover these constraints at the lowest possible cost, corresponding to the third approach from the list presented in the previous section.

In such situations the decision about which objective to use is often well-defined by the problem owner, who has a single interest in the problem, and other stakeholders may have other interests/objectives, that are not considered. Chapter 3 introduces a problem, which in fact attempts to include objectives from other stakeholders, and advocates that the results of such an optimisation may put the service provider in a better position for negotiation.

1.3 Metaheuristics

Both of the main problems that will be treated in this thesis are of a complexity which makes it unlikely that instances of real-life size can be solved optimally. For such cases heuristic solution approaches are a popular choice. Heuristics can be applied either by designing a problem-

specific heuristic for the problem at hand, or by adapting an existing metaheuristic approach. The work of this thesis will be focused on application of metaheuristics rather than specialised heuristics, and this section will give a brief presentation of some commonly used metaheuristics that are relevant in this context. A thorough description of a wide variety of metaheuristics can be found in Glover and Kochenberger [49], and reviews of the use of metaheuristics in vehicle routing can be found in for example Gendreau et al. [43], Cordeau et al. [19] and more recently Gendreau et al. [46].

Local search metaheuristics work by considering a starting solution, and gradually modifying this solution to improve it. At each iteration the neighbourhood of the current solution is examined, evaluating solutions that are similar to the current one. The neighbourhood is constructed by applying a minor modification, a *move* or *operator*, to the current solution. In a routing context such a move could typically consist of moving one customer to a different route (VRP), or swapping the positions of two or more visits (TSP or VRP). Other examples include the classical 2-opt, k-opt and Or-opt moves (cf. e.g. Jünger et al. [63]), for the TSP and VRP. For more involved problems the construction of successful neighbourhoods can be quite complex, since several problem constraints may impact how new solutions must be constructed in order to be feasible.

When implementing a metaheuristic one should also consider whether or not to allow intermediate infeasible solutions as part of the solution process. This is typically done by adding a term to the objective function which penalises the infeasibility, and can help the algorithm search more freely, and reduce the risk of getting caught in a certain part of the search space. Naturally the acceptance of infeasible solutions should still be somewhat limited, to ensure that feasible solutions to the problem are also discovered, and that the infeasible solutions still remain "close to" feasible.

The progress of a metaheuristic can often be divided into one or more phases of alternating intensification and diversification. Intensification are the phases where a good solution has been located, and work is focused on the immediate vicinity of this solution, in order to discover a similar and even better solution. Diversification are the phases that follow intensification, where an area has been searched thoroughly, and it is desirable to again broaden the search, to discover new promising solution regions. The key to a well-functioning metaheuristic often lies in a well-balanced mix between intensification and diversification. For algorithms that use diversifying deteriorating/hill-climbing moves, the best solution encountered may not coincide with the final current solution, and it is therefore recommended to keep a record of the best solution seen.

Finally it should be mentioned that apart from the specific metaheuristics that will be mentioned in the remainder of this section, a endless number of metaheuristics can be obtained by combining features of these and other existing metaheuristics, and such combinations have become

increasingly popular in recent years..

1.3.1 Tabu Search

Tabu Search (TS; introduced by Glover [48]; see e.g. Gendreau [39] for an introduction) is one of the most well-known and renowned metaheuristics, and has proven successful on a variety of routing problems over the years.

The basic idea of tabu search is in each iteration to select the best solution from the neighbourhood, and use memory of previous solutions to prevent the algorithm from cycling or being trapped at local optima. A fundamental concept is that of the *tabu list* which records characteristics of the latest visited solutions or latest applied moves. These operations are then declared *tabu* and forbidden as long as they are in the list, to prevent recent changes from being immediately undone. At each iteration the best non-tabu neighbour is chosen, even if it is not as good as the current solution, which means that local minima can be escaped. Sufficient information about the change/new solution is then recorded in the tabu list, where it remains for some number of iterations to ensure that a deteriorating move is not immediately reversed. An important decision when applying tabu search includes the determination of which characteristics of a move or a solution should be recorded in the tabu list.

Over the years, a number of additional features have been developed, which can be added to the tabu search algorithm to strengthen it further. These include such ideas as the use of variable tabu list length (variable tabu tenure), variable neigbourhood size, applying probabilities for different, and otherwise deterministic, parameters of the algorithm (probabilistic tabu search), and choosing the first improving or a random solution instead of searching the entire neighbourhood at each iteration. Gendreau [39] provides a list of references for more advanced extensions of tabu search.

1.3.2 Simulated Annealing

Simulated Annealing (SA; introduced by Kirkpatrick et al. [66]; see e.g. Henderson et al. [59] or Suman and Kumar [106] for an introduction and survey) is another well-tested metaheuristic, that has been around for more than 20 years.

The approach of simulated annealing is to occasionally accept deteriorating solutions, with a probability depending on the degree of deterioration. Simulated annealing typically considers a random neighbour as a candidate at each iteration. If this neighbour leads to an improvement of the current solution it is accepted for the next iteration. If the candidate solution is not improving, it is selected, despite this fact, with a

probability that depends on the deterioration of the objective value, and on the progress of the algorithm. At early stages the algorithm is more likely to accept deteriorating solutions to add diversification, whereas the probability drops towards the end, leading to an intensification of the search. The probability is controlled by the so-called *temperature*, which is reduced regularly throughout the computations. The start and end temperatures are parameters of the algorithm, and the end temperature is often used as the termination criterion.

The acceptance probability is usually expressed as an exponential function $e^{\frac{f(x)-f(x')}{T}}$, with $f(x)$ being the objective value of the current solution and $f(x')$ the objective value of the candidate solution. An often used temperature reduction function is $T_{i+1} = c \cdot T_i$, where T_i is the temperature at iteration i, and a suitably small value of $c \in (0, 1)$ is used.

Suman and Kumar [106] present a range of implementation choices, such as initial temperature and various cooling schedules suggested in the literature. In particular the choice of temperature range to use depends on the order of magnitude of the objective value for any given problem, and thus the range to be used must be adjusted for the problem instance at hand. Variations of the algorithm that have been suggested to improve the results obtained by simulated annealing, include different cooling schedules (either simpler or more complex), reheating, and various adaptive features to strengthen the otherwise memoryless procedure.

1.3.3 Variable Neighbourhood Search

Variable Neighbourhood Search (VNS; introduced by Mladenović and Hansen [80]; see e.g. Hansen and Mladenović [57] for a recent description) is based on the idea of combining several neighbourhoods to escape local optima, exploiting the fact that a solution that is a local minimum for one neighbourhood is not necessarily a local minimum when considering another neighbourhood, thus a change of neighbourhood can often be used to escape a local minimum. The VNS is based on an ordering of the available neighbourhoods, from smaller to larger, such that the simplest neighbourhood is used more often, and the more complex neighbourhoods are used only when the simple ones fail to find improving solutions. Neighbourhoods used together in VNS are often nested, such that the smaller neighbourhoods are subsets of the larger ones, or they may use a parameterised neighbourhood of increasing size.

The basic idea of the VNS is based on the ordering of the neighbourhoods. At each iteration one neighbourhood is considered and a local search procedure is performed. Whenever an improving solution is found, the next iteration will return to the simplest neighbourhood. Otherwise the search will proceed with a more complex neighbourhood. The procedure of the basic VNS performs a random move to *shake* or disturb the solution initially in each iteration, and then applies the local search

procedure to the shaken solution. The result of this local search is then considered the candidate solution, and is accepted if improving. The choice of neighbourhood for the next iteration depends on the outcome of the local search. The intermediate local search that is applied can also be used to construct a nested VNS procedure.

Different variations of the VNS have been suggested, for example applying some of the characteristics of simulated annealing, only checking a random neighbour from the neighbourhood in each iteration, or accepting non-improving solutions with a certain probability.

The idea of using a combination of several neighbourhoods in an algorithm will be revisited at a later point in this thesis, though the VNS framework as such will not be used.

1.3.4 Large Neighbourhood Search

Large Neighbourhood Search (LNS) as introduced by Shaw [104] is also a more recent metaheuristic, that has shown promising results on routing problems (e.g. Røpke and Pisinger [96]). It simplifies the use of neighbourhood moves as used by many other metaheuristics, by instead using a two-stage process at each iteration. First one or more *destroy* operators are used to remove a part of the current solution, and next one or more *repair* operators are applied to reinsert the removed elements into the solution. This enables the algorithm to apply a larger neighbourhood at each iteration, hence the name. The use of the destroy and repair operators in combination to construct the neighbourhood, means that these operators can often be simpler than the operators used for smaller-scale neighbourhoods, and the algorithm has shown good performance even with use of quite simple operators.

When elements are removed from the solution it is desirable to remove several similar elements in the same iteration, to increase the chance of obtaining fruitful reinsertions. If all of the removed elements are completely unrelated, there may only be one feasible position for each reinsertion, namely the same position the element was previously removed from. Instead it is desirable to remove related elements, to increase the chance of being able to swap some of the elements at insertion. In order to increase the likelihood of removing related elements from the solution, Shaw has suggested calculating a measure of relatedness when determining which elements to remove, such that each iteration would see the removal of a set of related elements, and would therefore provide the foundation of a potential benefit when these elements are reinserted. Furthermore the number of customers to remove at each iteration, and thus the amount of change made to the solution, is an important parameter that can be varied over time, and increase when diversification is needed, ressembling the idea of VNS. The other main issue for an implementation of LNS is the reinsertion procedure, which determines which solution element to insert next, and where to insert it. Finally, an

acceptance criterion for new solutions must be established – Røpke and Pisinger [96] have suggested a randomised acceptance criterion inspired by simulated annealing.

1.3.5 Iterated Local Search

Iterated Local Search (ILS; Lourenço et al. [76]) is not a "complete" metaheuristic in itself, rather it is a framework that can be applied, using some existing local search heuristic as a (potentially black-box) tool, and as the name suggests provides guidelines for iterating this local search heuristic. ILS is based on repeated restarts, which happen whenever a local optimum has been located by the local search procedure. Based on the solution given in this local optimum, a different solution is generated by use of some shaking/perturbation function, and the local search procedure is applied to this new solution, in the hope of locating a different and possibly better local optimum.

An implementation of ILS depends on four components: the initial solution, the local search procedure, the perturbation procedure, and the acceptance criterion, with the initial solution playing the least important role as computational time increases. These components can to some extent be considered separately, but ultimately also the interaction between them should be considered. In particular the perturbation function should be selected such that the procedure is able to escape local minima, while still preserving sufficient information about the latest good solution.

1.4 About this thesis

The main contribution of this thesis consists of the introduction of two new interesting problems to the research community, and suggesting initial solution approaches to each of these problems. Both problems integrate aspects that have traditionally been handled individually, such as routing and loading in freight transport, and timetabling and vehicle scheduling in passenger transport. Thus the two problems follow the trend of recent years, of considering increasingly integrated and complex problems.

The double TSP with multiple stacks (DTSPMS) presents a new take on the well-known pickup and delivery problem, by introducing a new way of handling real-life loading conditions. Pickup and delivery problems with LIFO (Last-In-First-Out) loading constraints have been studied previously, but the presence of several parallel LIFO stacks is new. Judging by the reception so far this is a problem that appeals to many researchers, due to its apparent simplicity. For the DTSPMS, a selection of classical metaheuristics have been implemented and tested, using two new feasible local neighbourhood operators. Additionally, a large neighbourhood

search algorithm has been applied, using some well-known, existing operators. Regarding exact solution approaches to the DTSPMS, different mathematical models based on branch-and-cut have been developed and examined. Furthermore, a set of test instances has been developed, on which some observations have been made regarding the impact of the problem size/configuration, and different bounds for the problem have been compared. Finally, some initial considerations have been made regarding the extension of the DTSPMS to the multiple vehicle case (DVRPMS), and an extension of the existing large neighbourhood search heuristic has been performed to provide some solutions to this problem.

The simultaneous vehicle scheduling and passenger service problem (SVSPSP) constitutes a new combined approach to vehicle scheduling/timetabling problems, integrating two problems which have typically been treated separately. This approach permits minor modifications of the timetable during the vehicle scheduling process, allowing increased flexibility for the solution of the vehicle scheduling problem, at the same time as passenger inconvenience can be considered. The SVSPSP is solved using a large neighbourhood search approach, and it is demonstrated that the solution of the integrated problem may indeed lead to improvements over the currently operated solutions.

1.4.1 Structure and reading guide

The thesis is primarily based on three research papers, which can be found in Appendices A, "The Double Travelling Salesman Problem with Multiple Stacks – Formulation and Heuristic Solution Approaches", B, "Exact Solutions to the Double Travelling Salesman Problem with Multiple Stacks", and C, "The Simultaneous Vehicle Scheduling and Passenger Service Problem". These papers represent the bulk of the work that has been carried out for this Ph.D. In addition to the papers, the first four chapters of the thesis provide supplemental and background material, making these chapter somewhat longer than what is traditional for a paper-based thesis. For each of the accompanying papers there is a section in the introductory report that contains further comments or details regarding the paper, such that each paper has a natural anchor in the main text of the thesis. When reading the thesis one may choose to follow this structure, and read each paper at the point that is suggested in the report. However, since all papers are written for publication and are therefore self-contained, it is naturally also possible to read the papers independently at any earlier point.

The research that is presented in this thesis can be divided into two main problem areas: The Double Travelling Salesman Problem with Multiple Stacks, and The Simultaneous Vehicle Scheduling and Passenger Service Problem. Following the general introduction to the area of multimodal transportation and other areas of general interest to the entire thesis in Chapter 1, the DTSPMS is covered by Chapter 2, combined with the

papers of Appendices A and B. The SVSPSP is covered in Chapter 3 and Appendix C, and finally some general concluding remarks are given in Chapter 4.

Chapter 2

The Double Travelling Salesman Problem with Multiple Stacks

This chapter covers the work on the Double Travelling Salesman Problem with Multiple Stacks (DTSPMS) that has been carried out in connection with this project. First a general introduction to the problem is given, discussing its relationship with other problems already known from the literature, and making some observations regarding properties of the problem. Then follow the two research papers that have been written on the DTSPMS, with some additional comments accompanying each paper. Finally some thoughts are given on how the DTSPMS can be generalised by including aspects from other problem classes, in particular a multi-vehicle variation is considered.

The Double Travelling Salesman Problem with Multiple Stacks (DTSPMS) is concerned with finding the shortest pair of routes performing pickups and deliveries in two separated regions. The items to be transported must be placed in one of several rows (horizontal stacks) in a container, such that each row maintains the LIFO (Last-In-First-Out) principle, while there are no mutual constraints between the rows. The problem permits neither intermediate repacking nor vertical stacking. An illustration of a small problem can be seen in Figure 2.1, which shows the two separated graphs with one depot and six customer nodes each, and a loading container with six positions organised in two rows.

The problem is defined on a pair of unconnected graphs – one for pickups and one for deliveries. The input consists of a set of orders, each one requiring transportation of one unit from a point in the pickup region to a point in the delivery region, i.e. each order includes a pickup location and a delivery location for one item. The items to be transported can

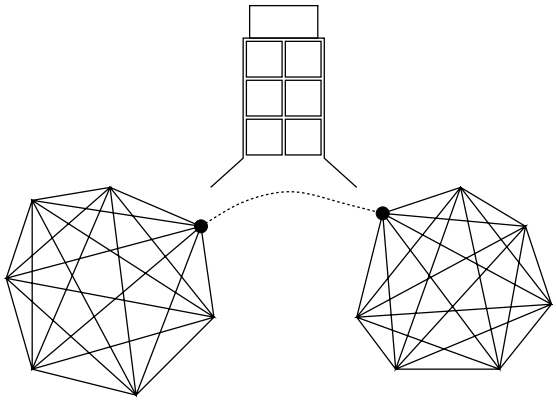


Figure 2.1: A small DTSPMS.

be seen as boxes or pallets of identical dimensions and each of the two regions has a depot. All pickups and deliveries must be carried out using the same container – this container is organised in several rows. The container is rear-loaded, and thus each row can be accessed as a LIFO stack. An item i is positioned in the container at pickup at the top/end of a given stack, and can only be delivered from this position. Since no repacking of the container is allowed, this means that all items that are picked up after i and are assigned to the same stack must be delivered before i can be delivered. The problem to be solved consists of determining the shortest Hamiltonian tour through each of the two graphs, in such a way that a feasible loading plan exists. Here, no time windows are considered, and the problem is only considered in its static version. Additionally, only unit orders are assumed at this point; orders containing multiple items can be treated as separate orders. All orders must be served, i.e. rejection is not allowed. The DTSPMS is NP-hard since it is a special case of the TSP, which is NP-hard [38].

The situation described above can occur in practice when a container is loaded onto a truck to perform the pickup operations, then returned by that truck to a local depot/terminal where the container is transferred onto a train, ship, or similar, to perform the long-haul transportation to the delivery region. Upon arrival at the depot/terminal in the delivery region, the container is again transferred to a truck, which carries out the deliveries. The terminals only have facilities to perform container movements, and do not offer any opportunities for opening and repacking of the container. Intermediate repacking may be impossible in real life if the transported goods are for example heavy, fragile, or hazardous, or there may be union regulations preventing the driver from handling the goods.

The project that forms the basis of the work on the DTSPMS that is described in this thesis, was initiated by a Danish software company, Easy Cargo Systems A/S, producing targeted software solutions for small and medium-sized companies in the transportation industry. The DTSPMS

had been encountered during talks with a potential customer, and was recounted as such. The problem seemed intriguing in its deceptive simplicity and thus the work presented in this chapter was commenced, as the problem did not appear to have been treated previously in the literature. Subsequently it has unfortunately been impossible to locate the actual real-life application of the problem, which could have been interesting for two reasons: 1) access to real-life data and comparison of the obtained solutions, and 2) rooting the problem in a real-life application. In particular the comparison of solutions would have been interesting, since it is certainly very difficult to construct a good manual solution, and it would have been interesting to know what kind of solutions an experienced planner would produce. However, even without the real-life connection the problem presents some interesting challenges, and will still be of value to the research community.

The DTSPMS is related to several other problems known from the literature. In particular, one should be mention the well-known Travelling Salesman Problem (TSP; cf. e.g. Applegate et al. [3]) which consists of finding the shortest possible Hamiltonian tour in a graph, visiting all nodes at the lowest possible cost. The TSP can be extended to the more general Vehicle Routing Problem (VRP; cf. e.g. Toth and Vigo [109] or Golden and Raghavan [51]), which can use several vehicles to accomplish the task of visiting all nodes at the lowest possible cost. All trips must then start at a depot where all vehicles are located. In particular the term Capacitated VRP (CVRP) is used for the VRP when each customer has a demand of a given size, and each vehicle has a fixed capacity that cannot be exceeded.

The DTSPMS is a variation of the vehicle routing problem with pickup and delivery (VRPPD), more specifically of its single vehicle variant, SVRPPD. The VRPPD typically occurs in courier services and a variation of it occurs in transportation of people (the Dial-A-Ride Problem, DARP).

The term DARP is often used for VRPPDs where the transported "items" are persons, and where customer inconvenience typically plays a role: in the objective function, by the introduction of ride time constraints, or by the presence of time windows. Furthermore the DARP often occurs in dynamic contexts, where not all data are known beforehand. As the LIFO principle is inherently "unfair", it most likely has no applications in a passenger transportation context (unlike the FIFO principle), and hence the band between the DARP and the DTSPMS is considered so thin that the DARP will be disregarded throughout the majority of this thesis.

The problem referred to here as the single vehicle routing problem with pickup and delivery (SVRPPD) is referred to by several different names throughout the literature, e.g. single vehicle pickup and delivery (SVPDP, [21]; SPDP, [85]), travelling salesman problem with pickup and delivery (TSPPD, [12]), and 1-VRPPD [28]. Here the name SVRPPD will be used

as the (hopefully) least confusing alternative, since the term TSPPD has also been used to describe a problem with simultaneous linehauls and backhauls (e.g. [42]), and PDP to describe a more general class of problems, of which the SVRPPD is only a subclass (by e.g. Cordeau et al. [20]). However, in the literature treating the SVRPPD with LIFO loading, the terms TSPPDL and TSPPD have been used, and therefore these names will be used when discussing the TSPPDL, rather than SVRPPDL, to ease the transition to the related literature.

The general VRPPD consists of a number of orders, each including a pickup point, a delivery point, a demand, and possibly time windows for the pickup and delivery operations. A fleet of vehicles (homogenous or heterogenous) with a given capacity are available to carry out the operations. The problem consists of finding a set of routes for the vehicles, such that all orders are served at the lowest possible cost (usually shortest driven distance). To this end, each pickup and delivery point must be visited exactly once, and each pair of associated pickup and delivery points must be visited by the same vehicle, with the pickup point being visited first. The maximum capacity of each vehicle can never be exceeded, and when time windows are present each customer must be visited within its allowed time window. The number of vehicles to be used may be fixed, or may be minimised as part of the objective function.

For clarity a mathematical model of the VRPPD is given below in (2.1)–(2.10), which is adapted from Cordeau et al. [20]. The model uses a fixed number of vehicles, and is shown with unit order loads, and without time windows and service times. For a mathematical formulation of the VRPPDTW the reader is referred to e.g. Cordeau et al. [20].

This model uses three sets of variables: x_{ij}^k indicates if the arc from i to j is travelled by vehicle k, t_i^k is the time where vehicle k visits node i and q_i^k is the load of vehicle k when leaving node i. $P = \{1, \ldots, n\}$ is the set of pickup nodes, $D = \{n+1, \ldots, 2n\}$ is the set of delivery nodes, and 0 and $2n+1$ are the depot nodes, initial and final respectively. The complete set of nodes in the graph is then $N = P \cup D \cup \{0, 2n+1\}$. Each arc (i, j) of the graph can be travelled at cost c_{ij} with duration τ_{ij}. The

model can then be expressed as follows:

$$\min \sum_{k\in K}\sum_{i\in N}\sum_{j\in N} c_{ij}x_{ij}^k \tag{2.1}$$

$$\text{s.t.} \quad \sum_{k\in K}\sum_{j\in N} x_{ij}^k = 1 \qquad i \in P \tag{2.2}$$

$$\sum_{j\in N} x_{ij}^k = \sum_{j\in N} x_{n+i,j}^k \qquad i \in P, k \in K \tag{2.3}$$

$$\sum_{j\in N} x_{ji}^k = \sum_{j\in N} x_{ij}^k \qquad i \in P \cup D, k \in K \tag{2.4}$$

$$\sum_{j\in N} x_{0j}^k = \sum_{i\in N} x_{i,2n+1}^k = |K|, \qquad k \in K \tag{2.5}$$

$$t_j^k \geq (t_i^k + \tau_{ij})x_{ij}^k \qquad i,j \in N, k \in K \tag{2.6}$$

$$t_i^k + \tau_{i,n+i} \leq t_{n+i}^k \qquad i \in P, k \in K \tag{2.7}$$

$$q_j^k \geq (q_i^k + u_j)x_{ij}^k \qquad i,j \in N, k \in K \tag{2.8}$$

$$q_i^k \in \{0,\ldots,Q^k\}, \quad t_i^k \in \mathbb{N}_0 \qquad i \in N, k \in K \tag{2.9}$$

$$x_{ij}^k \in \mathbb{B} \qquad i,j \in N, k \in K. \tag{2.10}$$

Constraint (2.1) is the objective function, expressing the total cost of all travelled arcs. Constraints (2.2) express that each pickup node must be visited, and (2.3) that each pair of pickup and delivery nodes must be visited by the same vehicle. Constraints (2.4) ensure flow balance for all customer nodes, and (2.5) ensure that each route starts and ends at the depot. The constraints (2.6) calculate the arrival time at each node (non-linear as given here, but can be linearised for implementation), and (2.7) use the arrival times to ensure that each pickup node is visited before its corresponding delivery node. (2.8) update the load variables after each visit, with the unit loads

$$u_i = \begin{cases} 1 & i \in P, \\ -1 & i \in D, \\ 0 & i \in \{0, 2n+1\}. \end{cases}$$

Finally, (2.9)–(2.10) express the domains of the variables, with Q^k being the capacity of vehicle k.

It should be noted that the precedence relation between a pickup node, i, and its corresponding delivery node, $n+i$, can also be expressed without the use of time variables, due to Ruland and Rodin [98], however further details of this approach will not be discussed here, as this precedence relation is not a complication in the context of the DTSPMS.

The TSPPDL is a extension of the SVRPPD, where all loading operations must adher to a LIFO principle, i.e. whenever the vehicle contains more than one item, the items must be delivered in the order opposite of their pickup order.

When comparing the DTSPMS and the TSPPDL, two major differences can be observed. In the TSPPDL (and VRPPD) it is important to maintain the internal ordering of any pickup/delivery pair, such that each pickup is performed before the corresponding delivery. This ordering is automatically observed in the DTSPMS, since the two graphs are separated and all pickups are completed before the first delivery takes place. On the other hand the DTSPMS provides multiple LIFO stacks for loading, whereas the TSPPD with LIFO loading provides only one stack, thus the crucial decision of which loading stack to use for each item only occurs in the DTSPMS.

Furthermore, vehicle capacity is not an issue in the DTSPMS as opposed to the SVRPPD. In the DTSPMS all items will necessarily be carried simultaneously during the transportation between the pickup and delivery regions, meaning that checking the capacity is trivial, and the problem will be infeasible if the vehicle capacity can be exceeded. In the (S)VRPPD it is necessary to check the capacity constraints after each pickup operation. In the DTSPMS, when considering each row individually, it does indeed have a maximum capacity, and the row capacity must be observed. Since all items are carried simultaneously at the intermediate long-haul, it suffices to check the row capacities here, rather than at each pickup operation. If the DTSPMS is extended to include multiple vehicles, naturally the row capacity and vehicle capacity will both have to be checked. The vehicle capacity will affect whether a given order can be accepted by a vehicle, whereas the row capacities will only affect the loading and thereby the routing of the vehicle.

The DTSPMS is symmetric in the sense that the loading rows are identical, and swapping two loading rows of a solution will produce another solution which is identical in any aspect considered here. However if, for example, loading stability were to be considered an issue, this would obviously no longer be the case. At the time being this symmetry provides some freedom for the driver to (slightly) improve loading stability when carrying out a route.

Many solution approaches to the SVRPPD focus on ensuring that each pickup customer precedes its corresponding delivery customer, which is not an issue in DTSPMS, since the pickup and delivery graphs are separated. In particular many valid inequalities for the regular TSPPD/TSPPDL such as listed in e.g. Cordeau et al. [21] and Cordeau et al. [23] are often trivially valid for the DTSPMS for this reason.

The DTSPMS shows some ressemblance to several other problems known from the literature. Such problems include other types of pickup and delivery problems, and various routing problems which incorporate loading constraints. These problems, as well as their similarities and dissimilarities with the DTSPMS, will be discussed in connection with the relevant literature in Section 2.1.

Finally it should be mentioned that there is some variation in the use of

the term "pickup and delivery problem" throughout the literature. In the most general sense it is being used to cover any problem that deals with both pickup and delivery operations, including for example backhaul or single-commodity problems (e.g. in the recent surveys by Berbeglia et al. [6] and Parragh et al. [85]). More narrowly it has been used to describe different subclasses that involve both pickup and delivery operations, e.g. problems with backhauls, or VRPPD type problems, which consider a range of orders, each requiring the transportation of goods between two given points (e.g. Savelsbergh and Sol [100] and Røpke et al. [97]).

Recent consensus (Cordeau et al. [20], Berbeglia et al. [6], Cordeau et al. [21], Parragh et al. [85]) seems to be on mainly using PDP in the broader sense, and using the term VRPPD (VRP with Pickups and Deliveries) or one-to-one PDP to refer to the more narrow problem class.

2.1 Literature and Related Problems

No research appears to have been done on the Double TSP with Multiple Stacks prior to this Ph.D. project, however a number of publications exist on various specialised topics in the bordering areas of the *Travelling Salesman Problem*, *Pickup and Delivery Problem*, and various types of *loading problems*, which all show similarities in different aspects. During the course of the current project, preliminary results have been presented at several conferences, which has spurred other researchers to commence work on the DTSPMS. In particular the paper by Felipe et al. [34] will be discussed in the context of the first paper A in Section 2.3.9.

For problems regarding the TSP or VRP with loading constraints, a number of different characteristics can be considered when describing a given problem:

- are the loading constraints in two or three dimensions?
- are the objects rectangular? Are they identical in shape and size?
- can the objects be rotated?
- can the objects be stacked?
- are several objects per customer allowed (and if so, how is this handled, usually split deliveries are not allowed)?
- are there multiple capacity measures (e.g. weight and space)?
- must all orders be served?

The first four points in the list are concerned with the packing aspects of the problem, while the last three points are concerned with routing issues, and each of these issues occur naturally in different applications. For the DTSPMS, each of these questions can in fact be answered by the answer that leads to the simplest problem, and yet the DTSPMS is no simple problem. However these questions illustrate a number of issues that arise in other routing/loading contexts, and some of them arise in the related problems that will be discussed below. In particular, this

section will not cover pure routing problems, like the regular TSP and (C)VRP, or pure packing problems, such as the knapsack or bin packing problems.

2.1.1 Surveys on Pickup and Delivery Problems

Several recent surveys exist on various areas within pickup and delivery problems, though to the author's knowledge, no dedicated surveys exist on problems more generally combining routing and loading.

The earliest work on pickup and delivery problems was done on the dial-a-ride problem, however due to the significant differences between the DARP and the DTSPMS, this section will mainly focus on work regarding pickup and delivery problems in a freight transportation context, and thus not cover the DARP, although a significant literature exists on this topic.

The first survey specifically stating pickup and delivery problems as its main focus area seems to be by Savelsbergh and Sol [100]. It presents the General PDP (GPDP), which is a VRPPD with added real-life complications, such as multiple requests with shared pickup or delivery locations, time windows for orders and vehicles, dynamic requests, and specified start and end locations for vehicles. The paper presents a mathematical formulation of this GPDP, discusses some defining characteristics of pickup and delivery problems (e.g. different common objective functions), and reviews early literature on the topic up to 1992. The VRPPD, DARP and VRP are mentioned as special cases of the GPDP. The survey of solution approaches divides these into static vs. dynamic, and next into single vs. multiple vehicle, and covers optimal methods as well as construction heuristics, but no iterative improvement heuristics.

Desaulniers et al. [28] give an introduction to the VRPPD and presents a model of the VRPPDTW, in a survey that focuses on static problem variants. The authors also comment on the role of service quality when the problem to be solved is a DARP. The literature up to 1999 is reviewed, covering heuristic and exact approaches, and finally an overview is given over different real-life applications where the VRPPD can be applied.

Cordeau et al. [20] cover the VRPPD and four of its application areas from the field of operational or "on demand" planning: dial-a-ride problems, urban courier service, dial-a-flight and emergency vehicle dispatch. Of these problems the urban courier service is usually dynamic in nature, whereas the remaining problems can be either static or dynamic. First the VRPPD is presented formally, and exact and heuristic solution approaches from the literature are reviewed for the single and multiple vehicle cases, before each of the four application areas are described in further detail, covering the problem characteristics and available literature. The authors also discuss the issue of multiple conflicting objectives

that often arises in the covered problems.

Berbeglia et al. [6] treat a broad class of problems where one or more commodities must be collected from and distributed to a set of customers. The paper provides a classification of these problems depending on the type of commodities, the distribution and mix of pickup and delivery customers, and the number of vehicles. Thus it covers several problem types that are not considered pickup and delivery problems in the more narrow sense. These include the 1-commodity PDTSP, where each customer supplies or demands a quantity of the same commodity (e.g. money), the VRP with backhauls, as well as the dial-a-ride problem and VRPPD. The paper also includes problems that allow the use of transshipment points, and where some customers may act as both pickup and delivery points.

According to the classification scheme provided by Berbeglia et al. [6] the SVRPPD and single vehicle DARP both match the same classification as the DTSPMS (1-1, P/D, 1). This implies that both consist of a set of requests which each include exactly one pickup and one delivery location (1-1), that no pickup and delivery locations coincide (P/D), and that only one vehicle is available (1). In the multiple vehicle case this becomes (1-1, P/D, m) which again coincides with the multi-vehicle DARP. This indicates that this classification unfortunately can not be immediately helpful in defining the distinction between these related problems.

[85, 86] by Parragh et al. constitute a two-part survey on pickup and delivery problems, of which the first considers the VRP with backhauls (VRPB), dealing with simultaneous distribution and collection, and the second covers problems where goods are transported between pairs of pickup and delivery locations, which is more interesting from a DTSPMS point of view. The survey suggests a classification scheme, presents a mathematical model of each of the problem classes that are covered, and surveys solution approaches from the literature. The solutions are divided into exact, heuristic and metaheuristic, and the size of the problems solved by the different paperse are summarised. Finally, a list of available benchmark instances is provided.

The classification scheme used by Parragh et al. [86] is based on common characteristics of the problems, but does not attempt to describe each class by a fixed set of fields, as Berbeglia et al. [6] do. This approach allows for a distinction between mixed and clustered/backhaul routes, i.e. whether or not linehaul and backhaul actions can be performed simultaneously or must be separated, and between the PDP and DARP problems. However the classification is still based on routing characteristics, and does not take such issues as loading constraints (e.g. LIFO) into consideration.

Recently, Cordeau et al. [21] have surveyed the latest advances within modelling and solutions of one-to-one pickup and delivery problems (thus excluding the many-to-many and one-to-many-to-one variations alto-

gether), covering both exact and heuristic approaches. The papers covered are divided into single and multiple vehicle problems, and the solution approaches into exact and heuristic. For each group a summary of the most significant recent contributions is given, with some details on valid inequalities for exact approaches, and on the algorithms used for heuristic approaches. This survey additionally provides a brief summary of a number of real-life applications where the PDP occurs.

2.1.2 Routing problems with LIFO loading constraints

The most significant similarities with the DTSPMS appear in the TSPPD with LIFO loading (TSPPDL), as treated in the papers by Carrabs et al. [12], Carrabs et al. [11], and Cordeau et al. [23]. In the TSPPDL a number of pickup and delivery requests must be served, all located within the same region, meaning that pickup and delivery operations are carried out simultaneously. In addition to the requirements regularly found for the SVRPPD, the TSPPDL implements a LIFO policy for the loading/unloading of the vehicle (i.e. for the vehicle as a whole, considering only one LIFO-stack). This means that at any given time the next operation can be either 1) a pickup of a new item, or 2) the delivery of the item that has been picked up most recently. If the pickup and delivery operations of the TSPPDL are separated the problem becomes identical to the one-stack DTSPMS, i.e. equivalent to solving a regular TSP.

The concept of *blocks* has been used when treating the TSPPDL (cf. Cassani [13], Carrabs et al. [12]), and can be used to provide further insight into the differences between the TSPPDL and the DTSPMS, arising from both the availability of multiple rows and the separation of pickup and delivery operations. A block $B(x^+, x^-)$ is a series of consecutive visits, beginning with a pickup customer x^+, and ending with its corresponding delivery customer x^-, such that it does not contain any pickup vertex without its corresponding delivery vertex, and vice versa. If any order were only partly contained, this would result in a so-called *cross*, as illustrated in Figure 2.2. A solution to the TSPPDL is infeasible if it

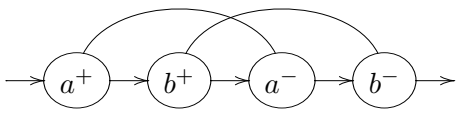


Figure 2.2: A cross, causing an infeasible solution.

contains any crosses. Blocks can be nested, and any TSPPDL solution can be broken down into a series of nested and sequential blocks, such that each request defines a valid block, as illustrated in Figure 2.3.

The notion of *crosses* can be used to illustrate the impact of the multiple rows that are available in the DTSPMS. In this case several colours/types of lines would be required for the illustration, to indicate different rows,

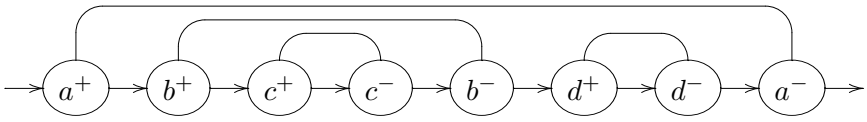


Figure 2.3: Illustration of blocks in the TSPPDL.

and a *cross* (and thus an infeasible solution) would then be found whenever two lines of the same type intersect, whereas lines of different type are allowed to intersect. The number of types of lines to be used for such a figure would be the same as the number of loading rows available in the problem. Figure 2.4 illustrates a solution that is feasible for the DTSPMS with two stacks but not for the TSPPDL, since the dotted and full lines intersect each other, but no two lines of the same type (dotted or full) intersect.

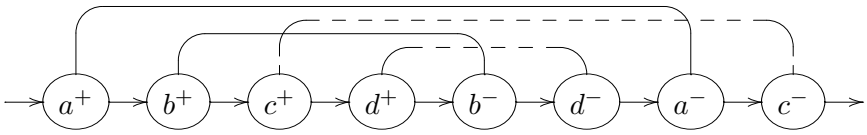


Figure 2.4: Illustration of valid blocks in the DTSPMS.

The recent literature on the TSPPDL consists mainly of a heuristic approach by Carrabs et al. [12], and exact approaches by Carrabs et al. [11] and Carrabs et al. [12]. Furthermore, related problems have been treated by Ladany and Mehrez [69] (double TSP with a single loading stack), Levitin and Abezgaouz [71] (TSPPDL), Erdoğan et al. [33] (TSPPDF, using FIFO instead of LIFO), Ficarelli [36], Cassani [13] (both TSPPDL, in Italian), and Pacheco [83, 84] (TSPPDL, in Spanish).

Carrabs et al. [12] produce heuristic solutions to the TSPPDL, using a variable neighbourhood search. The paper introduces two new local search operators for the TSPPDL, which are used in combination with four operators from Cassani [13], on test data that are based on TSPLIB[1]. Both Carrabs et al. [12] and Cassani [13] make extensive use of the concept of blocks when defining neighbourhood operators. The four operators introduced by Cassani [13] are based on exchange and relocation of orders or blocks. Carrabs et al. [12] extend the order-relocation operator to allow multiple relocations, and introduce another operator based on the well-known 2-opt operator (cf. e.g. [63]). The later uses the block concept to define a *reverse* procedure, which is necessary in order to apply any 2-opt like operator to a problem with pickups and deliveries, where simple reversion of a path would generally break the precedence of requests (pickup before delivery). The heuristic is tested on problems sized in the range 25 nodes (solved in <1 second) to 750 nodes (30–45 minutes). The results are compared to those of the VND heuristic by Cassani [13], which is outperformed.

[1] `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`

Carrabs et al. [11] solve the TSPPDL and TSPPDF to optimality, using an additive branch-and-bound approach, which allows for a combination of several lower bounds to be used, and has previously been applied to the TSP and the TSP with precedence constraints. The algorithm is used to combine the lower bounds originating from the assignment problem and the shortest spanning r-arborescence problem. Since these are lower bounds of the ATSP, which is in turn a lower bound of the TSPPDL, their use is combined with a set of filters to help impose LIFO constraints. Their approach is tested on TSPPDL instances containing 19–43 nodes. All instances of size up to 31 are solved to optimality within 3 hours, while 3 out of 9 instances of size 43 can be solved in the same time. For the TSPPDF the instances that can be solved are slightly smaller, with several instances of size 31 not being solved. This is caused by the different structure of the FIFO constraints, which weakens the applied filters.

In Cordeau et al. [23] three mathematical models for the TSPPDL are presented, along with different valid inequalities that are subsequently used to solve the problem using a branch-and-cut approach. Each model is an extension of a given model for the TSPPD. The first model associates a load variable to each node in the graph, stating the load of the vehicle when leaving the node, and requires a polynomial number of constraints added to the model. The second model instead attaches a load variable to each arc, expressing the load of the vehicle when travelling the arc, and also requires a polynomial number of constraints to be added to the model. The third model adds no new variables to the TSPPD model, but instead requires an exponential number of constraints to express the LIFO constraints on the existing variables. The computational experiments show that the third model obtains the best results, and solves all tested instances of size up to 38 nodes within an hour, and a few instances with 46 and 52 nodes.

As early as 1984, Ladany and Mehrez [69] interestingly presented a real-world problem, performing pickups and deliveries in two far-away regions. Having pickups and deliveries separated, their problem presents a similarity to the DTSPMS, that is not found in most other researched pickup and delivery problems. The problem involves a single LIFO stack for loading, and as a significant difference allows repacking of the container along the way, at a certain cost. The main difficulty of this problem lies in determining which repack operations to perform along with the routing, and a description is given of the calculation of the reshuffling costs. The typical stack size of the real-life problem is stated to be no more than 5, and hence the real-life problem is solvable by complete enumeration – a few comments are given regarding possible approaches to solving larger instances, but nothing more. A comparison is made between different reshuffling strategies for a small numerical example, and concludes that reshuffling is indeed worthwhile for the provided data.

Levitin and Abezgaouz [71] treat a problem with automated guided vehi-

cles (AGVs), that are used to transport materials between workstations in a factory. The materials are stackable, and an AGV can carry multiple loads simultaneously, working as a LIFO stack. The purpose is to construct a shortest possible route for the AGV. A feature of the problem is that a location can act as origin or destination of several requests, and that it is preferable to visit each location only once (this reduces total loading/unloading time). This means that if locations A and B both have loads destined for location C, A and B must be visited successively to avoid multiple visits to C. The paper states necessary conditions for the existence of a LIFO feasible route with only one visit at each station, and presents an algorithm for finding the shortest such route. A main focus of this problem is that of avoiding multiple visits, and thus grouping visits at origins with shared destinations, which removes this problem somewhat from the DTSPMS. The authors solve random instances with up to 100 nodes in solution times of less than one minute.

Finally, Erdoğan et al. [33] deal with the TSPPD with FIFO loading. This "reversed" loading policy provides a problem that exhibits fewer similarities to the DTSPMS, however the mathematical model presented for the TSPPDF can be changed to express the TSPPDL by changing a single constraint. In addition to the mathematical formulation of the TSPPDF, the authors present five local search operators and two different metaheuristic solution approaches (probabilistic tabu search and iterated local search) which use the presented operators. The algorithms are tested on the instances from Carrabs et al. [12] with sizes in the range 25-751 nodes, and the best approach shows to be the probabilistic tabu search, used in combination with a nearest neighbour based construction heuristic. The smaller instances are solved within a few seconds by this method, while run times go up to an average of more than an hour for the most difficult of the largest instances.

2.1.3 Routing problems with general loading constraints

Several papers deal with combined routing and loading problems, such as the capacitated vehicle routing problem with 2- or 3-dimensional loading constraints (2L-CVRP and 3L-CVRP). In this type of problems, a VRP must be solved such that a feasible loading plan exists for each vehicle. In this case the items are usually rectangular but of varying size, and must fit into a rectangular loading space in two or three dimensions without overlapping. Additionally each item must be accessible at its time of delivery, and each customer location should be visited exactly once, i.e. split deliveries are not allowed, and for any customer that requires more than one item, all items must be picked up/delivered during one visit. It is often assumed that items cannot be rotated, although some research has also been extended to non-rectangular objects with no fixed orientation.

Iori et al. [61] solve the 2L-CVRP to "almost certain optimality", using a branch-and-cut procedure. They first present a model for the problem which includes two sets of constraints that are non-polynomial in size, and then present separation procedures for these constraints. The two sets of constraints that are initially relaxed impose (a) the loading capacities of the vehicles and non-overlap of the loading plan (including the solution of a bin packing problem) and (b) that each customers can only be visited once (this also has implications for feasible loading plans), which is modelled using infeasible paths. The separation procedure for the bin packing problem is heuristic, meaning that it may, in rare cases, return an invalid cut. In these cases the solution is marked as not guaranteed optimal. 60 test instances are considered in the paper, of which the 45 smaller ones (up to 25 customers and 91 items) were solved to optimality within an hour, and 10 of the remaining 15 (up to 35 customers and 114 items) were solved within 24 hours.

Gendreau et al. [45] consider the capacitated vehicle routing problem with 2-dimensional loading constraints (2L-CVRP), and present a tabu search procedure to solve it. Two variations of the problem are considered: one in which the items of each customer must be directly available at delivery, and not be blocked by items belonging to other orders, and one in which such a restriction is not imposed. At each iteration a routing is determined, and then a heuristic is used to determine a corresponding loading. Loading infeasibilities are allowed at the cost of a penalty. Two intensification procedures are used, both making use of an improved loading heuristic. Of the problems where an optimal solution is known (16–36 nodes, up to 114 items), the algorithm matches this solution for 57% of the problems, and for instances where no optimal solution is known, the previously best found solution (found by branch-and-cut by Iori et al. [61]) is on average slightly improved. Running times are usually less than 5 minutes. Some larger instances with no reference solution are considered, using running times up to an hour. These instances are mainly used to consider the cost of loading restrictions, and examine some statistics of the constructed routes.

Gendreau et al. [44] study the 3L-CVRP, which is a natural extension of the 2L-CVRP. The packing used for the 3-dimensional problem requires considerations regarding stacking of items; some items may be fragile (limiting the stackability), sufficient supporting area must be provided for stacked items, and a LIFO loading policy is imposed. Furthermore, unlike the 2L-CVRP, the items are allowed to be rotated 90° around a vertical axis. The 3L-CVRP is solved using a tabu search approach, in which a loading problem is solved to determine feasibility of a solution. The loading problem is solved as a 3-dimensional strip-packing problem using tabu search, and the length of the loading is then compared to the length of the container. If necessary, both constructive and improvement heuristics are applied to the loading subproblem. The solution allows intermediate solutions that are infeasible with regard to loading weight or loading length, at a penalty cost. The algorithm is tested on instances

from the litereture as well as real-life instances. The first 27 instances are taken from the literature and have 15–100 customers, and 26–199 items. For all instances a feasible solution is identified, and for those instances where a simpler heuristic was able to produce an initial solution, this is on average improved by 40%. Running times are often less than half an hour, and never exceed two hours. On the real-life instances (44–64 nodes, 141–181 items) the algorithm is allowed running times of 1, 10 and 24 hours, and the solution value of the final solution improves the initial solution by 30–40%. By increasing the solution time from 10 to 24 hours the solution value are improved by 4% on average.

2.1.4 TSP/VRP with Backhauls

The term TSP/VRP with backhauls describes a class of routing problems where some orders must be delivered from a depot to the customers (linehaul), and other orders must be picked up from the customers and brought back to the depot (backhaul). This problem has been studied rather extensively in the literature, which also means that several different flavours of the problem have been covered: The orders may be served simultaneously or consecutively, and when served simultaneously there may or may not be customers requiring both types of service. Additionally the usual variations can apply, such as vehicle capacities, time windows and the opportunity of performing split deliveries. In the literature several different names are used for routing problems with backhauls, including "simultaneous pickup and delivery" ([81]) and "deliveries and collections" ([4]). An general introduction to the VRP with Backhauls can be found in Toth and Vigo [110].

In comparison to the DTSPMS, the TSPB obviously lacks the ties between pairs of pickup and delivery customers. The clearest connection appears when both are considered as "two TSPs with interconnected routing", where the connection for the TSPB can be viewed as two open-ended TSPs with a common endpoint.

The original form of this problem contained two distinct sets of orders containing linehaul and backhaul customers, requiring that all linehaul operations must be completed before the backhaul operations can be commenced. This problem has been considered in the TSP version by Gendreau et al. [40] and Gendreau et al. [41] using heuristic approaches, and in the VRP version by Toth and Vigo [107] (exact) and Toth and Vigo [108] (heuristic).

Additionally a version of the problem has been studied, where line- and backhaul customers can be served simultaneously, in the VRP-version by Montané and Galvão [81] and Mosheiov [82] (split deliveries allowed), and by Baldacci et al. [4], who found exact solutions to the TSP-version (100–150 customers, all instances solved in less than an hour).

Gendreau et al. [42], Duhamel et al. [31], Salhi and Nagy [99], and Grib-

kovskaia et al. [52] all consider heuristic solutions to problems where the operations are mixed and customers may require both pickup and delivery operations.

Finally the VRP version of the problem with time windows has been solved heuristically by Potvin et al. [94] and Zhong and Cole [115].

2.1.5 Other related TSP/VRP variants

In addition to the above mentioned problems, some less studied variants of the TSP and VRP exist, which contain loading, stacking or other additional constraints that affects which routings are allowed.

2.1.5.1 Multi-Pile Vehicle Routing Problem

The multi-pile vehicle routing problem (MPVRP) is a variation of the VRP with loading constraints, where the loading space is divided into several different (vertical) piles/stacks. The MPVRP was introduced by Doerner et al. [30] and has been further treated by Tricoire et al. [111]. The MPVRP is concerned with the delivery of goods to users, and has no pickup aspect. The goods are packed on pallets (of varying height) which gives rise to the multi-pile approach, however some goods do not fit on one pallet, and will instead extend across the full length of the vehicle, covering all piles. The problem consists in determining a set of routes with the smallest possible routing cost, such that for each route there exists a feasible loading of the vehicle. Additionally it is required that each customer can only be visited once, and repacking is not allowed, i.e. all items destined for a given customer must be grouped, such that they can all be delivered at the same time, without requiring intermediate handling of items meant for other customers. An illustration of how a loading could look for the MPVRP can be seen in Figure 2.5.

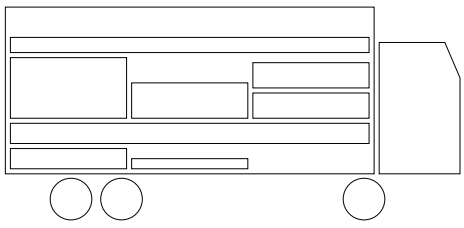


Figure 2.5: Loading for the MPVRP.

In Doerner et al. [30], the MPVRP is solved using tabu search and ant colony optimisation (ACO). The tabu search approach is based on a modified objective function that allows infeasible solutions by adding a penalty costs for overloaded vehicles, and additionally applies multi-starts. The ACO approach is based on a framework for the general VRP, with some modifications to cater for the loading restrictions. This includes an extra pheromone matrix for loading, where a pair of customers

is preferred if they take up a lot of loading space, and only lead to a small amount of space being potentially wasted. The loading feasibility subproblem is solved using a combination of a simple heuristic and a dynamic programming approach. The algorithms are tested both on instances derived from the CVRP literature, with instance sizes ranging from 50 to 199 nodes, and taken from a real-world application of the MPVRP with 76 nodes. A maximum of two hours solution time is allowed, but this limit is hardly ever reached. The ACO provides the better results for all but the smallest instances. Additionally, some results are presented on the potential of reducing the number of vehicles used in the solution, by adjusting a parameter of the objective function.

Tricoire et al. [111] use a Variable Neighbourhood Search (VNS) and a branch-and-cut approach to solve the MPVRP. Several different approaches are applied to solve the subproblem – one exact and one greedy heuristic, along with a heuristic from Doerner et al. [30]. Additionally a number of lower bounds from the literature are used on the subproblem. The test instances from [30] are used for the heuristic with 30 minutes run time. The VNS algorithm demonstrates better performance than the previous TS and ACO approaches, and often improves the previously best known solution. The branch-and-cut approach is tested on instances of reduced size, with 20–56 customers, and with a time limit of 2 hours. The VNS is used to produce initial solutions, and to produce a pool of proven feasible/infeasible routes to be used by the branch-and-cut algorithm. All tested instances with up to 38 customers are solved to optimality within the time limit, while only 1 instance of 44 customers, and no larger instances can be solved.

The similarity between the MPVRP and the DTSPMS lies in the availability of multiple stacks for the loading problem. However the stacks play a rather different role in the two problems, meaning that the similarities are fewer than they appear at first sight. In the DTSPMS the stacks affect the implications of the LIFO principle, whereas LIFO is not at all present in the MPVRP. In the MPVRP the stacks impose a complication on the search for capacity feasible loading plans, by requiring load to be spread over stacks for better usage, while capacity feasibility is not an issue in the DTSPMS.

2.1.5.2 Black and white TSP

The black and white travelling salesman problem (BWTSP) is concerned with finding a shortest Hamiltonian tour visiting all vertices in a graph whose vertex set is divided into black and white vertices. It is required that between two consecutive black vertices the solution contains no more than a certain number of white vertices, and the total distance between two black vertices can also not exceed a given limit. The BWTSP typically arises in telecommunications and in the airline industry (scheduling with maintenance requirements).

Bourgeois et al. [9] presents three construction heuristics which may produce infeasible solutions, one repair heuristic and one improvement heuristic. The first construction heuristic constructs a TSP solution, disregarding the special black-and-white constraints. The second and third construction heuristic both start out by constructing a TSP solution from the white vertices, and then inserts the black vertices. The repair heuristic takes as input a TSP tour where one or both of the special black-and-white constraints are violated, and attempts to make the solution feasible by moving black vertices into or white vertices away from problematic sequences. The improvement heuristic is based on the 2-opt procedure, modified to ensure that feasibility is not broken when a move is performed. The resulting heuristics are tested on 1080 randomly generated instances with 50, 100, and 200 nodes, and with a varying distribution of black nodes. All problems that were solved, were solved in less than 3 minutes, however for some problems no feasible solution could be found at all. The results show that the second approach was better at locating a feasible solution in difficult instances, whereas the two other approaches on average produced better results.

In Ghiani et al. [47] an integer linear model for the black and white TSP is presented, along with some valid inequalities for the problem, leading to a branch-and-cut algorithm. The presented valid inequalities are a combination of adaptations of known TSP inequalities, and new inequalities developed for the BWTSP. The latter can be split into two groups concerned with 1) the number of white vertices and 2) the distance between two consecutive black nodes. The algorithm is tested on a combination of randomly generated instances, and instances derived from TSPLIB. Within 10,000 seconds it produces optimal solutions to most instances with up to 60 nodes, and to some instances with 100 nodes.

A common trait between the BWTSP and the DTSPMS is that both involve some sort of distinction between the vertices that does not occur in the regular TSP. In the BWTSP this is the distinction between black and white vertices, and in the DTSPMS this is the distinction between customers whose orders are loaded in different loading rows. A significant difference is that in the BWTSP this division of the vertices is an exogenous variable (given beforehand), while in the DTSPMS this division is an endogenous variable (part of the decision to be made). These existing similarities between the BWTSP and the DTSPMS do not lead to any immediate insights, but the relationship could be investigated further.

2.1.5.3 VRP with cross-docking

The VRP with cross-docking (VRPCD) presents a different way of handling loading issues when multiple vehicles are present. In the VRPCD all vehicles return to the terminal/cross-dock after performing the pickup operations, and the goods are then redistributed between the vehicles be-

fore the deliveries are carried out. Thus reloading is permitted, to allow transfer of items between vehicles. Furthermore the actual repacking cost/time is considered, i.e. there is a preference for using the same vehicle for pickup and delivery for a given order. Loading ordering for the VRPCD does not seem to have been considered, for example by imposing some kind of LIFO principle, which could be justified by the fact that not all orders are in fact repacked. The treatment in the literature of the vehicle routing problem with cross-docking is still quite sparse.

The VRPCD tends to cluster customers with nearby locations in the same vehicle, and thus the solution becomes more difficult if several requests have nearby pickup locations but far apart delivery locations (in particular when reloading is costly). On the contrary, in the DTSPMS, the clustering of requests into rows does not relate as directly to geographical proximity.

Lee et al. [70] consider the VRPCD from a supply chain point of view. The authors require the simultaneous arrival of all trucks at the cross-dock after the pickup phase, to avoid vehicles having to wait before being repacked and departing. A tabu search algorithm is implemented to solve the problem. Instances with up to 50 nodes are considered, all of which are first solved to optimality using enumeration, to assess the quality of the heuristic solutions. Within 1000 iterations the tabu search procedure produces solutions on average 3.75% above optimum, however no running times are reported; it is only stated that the results are obtained in a "reasonable amount of time".

Wen et al. [112] use tabu search to solve the VRPCD, within an adaptive memory procedure (AMP) allowing the tabu search to perform restarts. The tabu search algorithm allows infeasible solutions with a penalty for exceeded capacity, duration and time windows, and variable neighbourhood size is used to intensify the search towards the end of the computation. Additionally, a initial bound on the cost of each neighbourhood move is used to determine if a move looks promising enough to calculate its cost exactly. The algorithm is tested on data sets based on real-life data with up to 400 nodes in total, at running times of maximum 5 minutes. On small instances (20 orders) the algorithm consistently arrives within 1% of the lower bound, whereas on larger instances (30–200 orders) it consistently arrives within 5% of a weaker lower bound.

Ideas from cross-docking could be applied to the DTSPMS, if it were to be solved with some degree of reloading allowed at the terminal/depot at a certain cost. The VRPCD and DTSPMS are not immediately very similar, however the underlying problem that both attempt to deal with are related, namely that of considering a trade-off between routing and loading costs. How much longer is it acceptable for the truck routes to be, if repacking should be avoided? The DTSPMS seeks the best routes possible when no repacking is allowed, and the cost of this solution can then be compared to the cost with no loading considerations. The VRPCD includes the cost and time of the repacking as a parameter, and

can present a lot of variation in the solutions by adjusting this parameter, as demonstrated in [112].

2.1.6 Applications

Several papers have been published which discuss real-life occurrences of routing problems with loading aspects or other complications, and the difficulties this poses. A selection of such papers will be presented in the following, giving a flavour of the complications that real-life situations may impose on routing and loading problems.

Shang and Cuff [103] solve a multiobjective pickup and delivery problem for transporting patients' records, equipment and supplies between a number of medical facilities. The problem involves time windows and some special characteristics, in particular the option of letting items transfer between vehicles in the field as part of the transport. The problem can be considered a DARP with some special properties (e.g. transfer opportunities and locations shared by several requests). Vehicle capacity is not an issue, nor is ride time, as long as ready times and due times are respected. The paper presents an interactive, constructive heuristic for the problem, which allows the user to accept or reject a given solution, and resolve with modified parameters until a satisfactory solution has been found. A subprocedure is used to construct clusters, which are then evaluated and inserted into the existing vehicle schedule. The paper handles the multiple objectives (number of vehicles used, travel time minimisation, and tardiness) on a "one objective at a time"-basis, optimising with regard to one objective and imposing a limit on the value of the others, or alternatively weighting them together. The algorithm is tested on real-life data with 300 requests (of which 133 have no time windows, and are inserted at the end of the procedure). The results produced are superior to the previously used cyclic, fixed-route solution used by the company, which does not allow use of the transfer opportunity (37% improvement for running time less than 4 minutes).

Xu et al. [114] describe the solution of a real-life pickup and delivery problem, with several complicating restrictions, including multiple time windows, multiple carriers, heterogenous vehicles, legal working hour constraints, LIFO constraints, and compatibility constraints between orders and carriers, vehicles and other orders. The cost function includes fixed costs, total distance, driving time and driver layover time. The problem is solved using a heuristic column generation procedure, where the routing and scheduling of each vehicle is solved as a subproblem, which contains all the complications. Two different heuristics have been implemented to solve this subproblem. Additionally a dynamic programming algorithm is presented, which can be used to produce lower bounds for the subproblems, but is only applicable for smaller instances, and is used to evaluate the heuristic results. The algorithm is tested on two sets of data, both of which are based on real-life data: medium sized instances

with 100–420 nodes, and large instances with 600–1000 nodes (both using a relatively large fleet of vehicles, with 2–5 orders being served by each vehicle). For the medium-sized instances the best heuristic gives results that are on average within 4% of the lower bound in less than 2 minutes. For the larger instances the algorithm providing the better results uses up to 7 hour of CPU time on average, while the other uses less than 45 minutes, and provides solutions that are on average within 5% of the better solutions.

Savelsbergh and Sol [101] consider a pickup and delivery problem with time windows, a heterogenous fleet, compatibility constraints, driver restrictions, multiple delivery points for each pickup, and 40% dynamic requests. The problem is modelled as a set partitioning model, and solved using a branch-and-price approach, with heuristics used to speed up the solution process, and some minor modifications to handle parts of the dynamics, cost structure, and driver regulations. The procedure is tested on randomly generated instances and on real-life instances. A simulation is performed to evaluate the quality of the solutions, and they are compared to the actual planners' decisions, showing potential cost reductions of 3–4%, by solving the reoptimisation problem hourly with a time limit of 5 minutes. The real-life data set contains around 90 vehicles and some 100–200 requests to assign at each run.

Finally, some researchers (e.g. Goel and Gruhn [50] and Hasle [58]) have used the term "rich VRP", which is a broad term covering all types of VRP problems with additional (real-life) constraints. The purpose of this is often to point out that the theoretical and "clean" problems typically considered in the literature (such as the regular TSP, VRP, and VRPTW) are inapplicable in practice, since the real world will often impose numerous additional constraints that do not easily fit into these often-studied problem classes. Such constraints include a variety of restrictions on loading, order types, fleet composition, driver regulations, dynamically incoming orders, and the need for fast response times.

Goel and Gruhn [50] present what is referred to as a General VRP, incorporating a variety of features from real-life applications. These include load acceptance (not all orders must be serviced), time windows, a widely heterogenous fleet of vehicles (capacities, travel times, costs, depots) compatibility constraints, orders split over different locations, and routing restrictions. The problem has been encountered in road feeder services for air freight. The problem is solved using a Reduced VNS approach (VNS with reduced neighbourhood search), and an LNS. To simulate a dynamic environment the heuristics are only allowed a few seconds of computation time, on randomly generated instances of 500–2500 total orders and 100–500 vehicles.

2.2 Properties of the DTSPMS

A mathematical formulation of the DTSPMS is provided in each of the papers A and B, and will be repeated here for completeness.

The DTSPMS is defined on two complete graphs $G^T = (V^T, E^T)$, where $T \in \mathcal{T}$ and $\mathcal{T} = \{\mathrm{P}, \mathrm{D}\}$ denotes pickup and delivery, respectively. The set of pickup nodes is $V^{\mathrm{P}} = \{v_0^{\mathrm{P}}, \ldots, v_n^{\mathrm{P}}\}$, where v_0^{P} is the depot in the pickup region and $v_1^{\mathrm{P}}, \ldots, v_n^{\mathrm{P}}$ are the pickup locations of the orders $1, \ldots, n$. The delivery graph is defined correspondingly. For each arc in the graphs the cost c_{ij}^T is given.

The problem can be formulated using three sets of binary variables, indicating the routing, the precedence relations between orders, and the assignment of orders to loading rows.

$$x_{ij}^T = \begin{cases} 1 & \text{if arc } (i,j) \text{ is used in graph } G^T, \\ 0 & \text{otherwise,} \end{cases}$$

$$y_{ij}^T = \begin{cases} 1 & \text{if node } v_i^T \text{ is visited before node } v_j^T \ (i \neq j), \\ 0 & \text{otherwise,} \end{cases}$$

$$z_i^r = \begin{cases} 1 & \text{if item } i \text{ is placed in row } r, \\ 0 & \text{otherwise.} \end{cases}$$

The model can then be stated as follows:

$$\min \sum_{\substack{T \in \mathcal{T} \\ i,j \in V^T}} c_{ij}^T \cdot x_{ij}^T \tag{2.11}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{V}^T} x_{ij}^T = 1 \qquad T \in \mathcal{T}, j \in \mathcal{V}^T \tag{2.12}$$

$$\sum_{j \in \mathcal{V}^T} x_{ij}^T = 1 \qquad T \in \mathcal{T}, i \in \mathcal{V}^T \tag{2.13}$$

$$y_{ij}^T + y_{ji}^T = 1 \qquad T \in \mathcal{T}, i,j \in \mathcal{V}_C^T \tag{2.14}$$

$$y_{ik}^T + y_{kj}^T \leq y_{ij}^T + 1 \qquad T \in \mathcal{T}, i,j,k \in \mathcal{V}_C^T \tag{2.15}$$

$$x_{ij}^T \leq y_{ij}^T \qquad T \in \mathcal{T}, i,j \in \mathcal{V}_C^T \tag{2.16}$$

$$y_{ij}^{\mathrm{P}} + z_i^r + z_j^r \leq 3 - y_{ij}^{\mathrm{D}} \qquad r \in \mathcal{R}, i,j \in \mathcal{V}_C^T \tag{2.17}$$

$$\sum_{r \in \mathcal{R}} z_i^r = 1 \qquad i \in \mathcal{V}_C^T \tag{2.18}$$

$$\sum_{i \in \mathcal{V}_C^{\mathrm{P}} \cup \mathcal{V}_C^{\mathrm{D}}} z_i^r \leq L \qquad r \in \mathcal{R} \tag{2.19}$$

$$x_{ij}^T \in \mathbb{B} \qquad T \in \mathcal{T}, i,j \in \mathcal{V}^T, i \neq j \tag{2.20}$$

$$y_{ij}^T \in \mathbb{B} \qquad T \in \mathcal{T}, i,j \in \mathcal{V}_C^T, i \neq j \tag{2.21}$$

$$z_i^r \in \mathbb{B} \qquad r \in \mathcal{R}, i \in \mathcal{V}_C^{\mathrm{P}} \cup \mathcal{V}_C^{\mathrm{D}}. \tag{2.22}$$

The objective function sums the cost of all used arcs in the graph. Constraints (2.12) and (2.13) are flow conservation constraints, stating that one unit of flow must enter and exit each node. Constraints (2.14) ensure that for each pair of nodes (i, j) a precedence variable must be set to one, i.e. either i is before j or j is before i. (2.15) ensure the consistency and transitivity of the y variables: that if i is before k and k is before j, then i must necessarily be before j. Constraints (2.16) make sure that if an arc (i, j) is used, then the precedence variable is set accordingly (i is visited before j, i.e. $y_{ij} = 1$). Constraints (2.17) express the LIFO constraints, that only apply when two items are in the same row r: if i and j are placed in the same row ($z_i^r = z_j^r = 1$), and i is picked up before j ($y_{ij}^{\mathrm{P}} = 1$), then j must be delivered before i ($y_{ji}^{\mathrm{D}} = 1$). Finally, (2.18) ensure that all items are assigned to exactly one row, and (2.19) enforce the row capacity L.

Pickup and delivery problems (in the narrow meaning of the term, i.e. problems of type SVRPPD) are traditionally modelled with a order index $i \in \{0, \ldots, 2n + 1\}$ for an instance with n orders, with $P = \{1, \ldots, n\}, D = \{n + 1, \ldots, 2n\}$ and depots 0 and $2n + 1$. However since the pickup and delivery graphs of the DTSPMS are completely separated, such a formulation did not seem natural, and did not improve the readability of the model. Instead, it has been chosen to define the variables as can be seen above, with one index used to indicate the order $i \in \{1, \ldots, n\}$, and another index to indicate whether the node is a pickup or delivery node $T \in \mathcal{T}$.

A dataset for the DTSPMS has been developed for this project and made available online[2]. The test instances have been generated randomly, by finding two sets of n random (real) points in a 100×100 square. The depot is placed in the centre of the square at $(50, 50)$. All distances are Euclidean distances rounded to the nearest integer, in accordance with the conventions from TSPLIB. This rounding implies that the triangle inequality is not preserved, however none of the algorithms presented here rely on this inequality.

2.2.1 Observations on bounds

Optimal solutions have been determined for all instances with 15 orders, and in the following a discussion will be given regarding the relationship between these optimal solutions and various bounds, as well as on similar bounds obtained for larger instances.

The values for some TSP bounds and optimal solutions split on the pickup and delivery graphs have been summarised in Table 2.1. The first column shows the instance number, followed by the value of the optimal DTSPMS solution z^*. Columns z_{P}^* and z_{D}^* show the value of the DTSPMS-optimal path through each graph. z^{TSP} shows the solution

[2] `http://www.transport.dtu.dk/datasets/DTSPMS`

Inst.	z^*	z^*_{P}	z^*_{D}	z^{TSP}	$z^{\mathrm{TSP}}_{\mathrm{P}}$	$z^{\mathrm{TSP}}_{\mathrm{D}}$	$\frac{z^*}{z^{\mathrm{TSP}}}$	$\frac{z^*_{\mathrm{P}}}{z^{\mathrm{TSP}}_{\mathrm{P}}}$	$\frac{z^*_{\mathrm{D}}}{z^{\mathrm{TSP}}_{\mathrm{D}}}$
R00	741	398	343	707	367	340	1.05	1.08	1.01
R01	754	378	376	720	378	342	1.05	1.00	1.10
R02	658	334	324	635	319	316	1.04	1.05	1.03
R03	768	380	388	733	377	356	1.05	1.01	1.09
R04	708	403	305	689	403	286	1.03	1.00	1.07
R05	737	373	364	716	371	345	1.03	1.01	1.06
R06	836	374	462	811	364	447	1.03	1.03	1.03
R07	690	382	308	651	364	287	1.06	1.05	1.07
R08	826	427	399	789	390	399	1.05	1.09	1.00
R09	768	359	409	751	349	402	1.02	1.03	1.02
avg.	749	380	368	720	368	352	1.04	1.03	1.05

Table 2.1: Lower bounds, 15 orders.

value of two individual TSPs, which is the lower bound referred to as nS (n-stack) in Paper A. $z^{\mathrm{TSP}}_{\mathrm{P}}$ and $z^{\mathrm{TSP}}_{\mathrm{D}}$ show the length of the optimal TSP tour in the pickup and delivery graphs respectively, thus $z^{\mathrm{TSP}} = z^{\mathrm{TSP}}_{\mathrm{P}} + z^{\mathrm{TSP}}_{\mathrm{D}}$. The column $\frac{z^*}{z^{\mathrm{TSP}}}$ shows the gap between the optimal solution and the nS bound, and $\frac{z^*_{\mathrm{P}}}{z^{\mathrm{TSP}}_{\mathrm{P}}}$ and $\frac{z^*_{\mathrm{D}}}{z^{\mathrm{TSP}}_{\mathrm{D}}}$ show the gaps between the TSP and DTSPMS optimal tours for each of the graphs P and D.

The numbers of Table 2.1 show that the n-stack bound, $\frac{z^*}{z^{\mathrm{TSP}}}$, is a reasonable bound for problems of moderate size (15 customers), with an average gap of the optimal solution of 4%. However as L increases, the accordance between the TSP and DTSPMS solutions decreases, and so does the strength of this bound. The two last columns show that the way to obtain a DTSPMS-optimal solution from a TSP-optimal solution can have different forms, and that the deviation from z^{TSP}_T is not necessarily evenly distributed between the pickup and delivery routes. For some instances the DTSPMS-optimal solution consists of one TSP-optimal route and one route which is rather far from its TSP-optimal (in particular R01, R04, and R08), and for other instances both routes have undergone some modifications compared to the TSP-optimal (such as R02, R06, and R07). For larger instances it should not be expected that very many DTSPMS-optimal solutions contain one route that is TSP-optimal, however it seems very likely that one route can be much further away from its TSP-optimal than the other.

Table 2.2 shows some upper bounds that are based on the lower bounds in Table 2.1. $z^{\text{TSP-D}}_{\mathrm{P}}$ ($z^{\text{TSP-P}}_{\mathrm{D}}$) shows the length of the optimal DTSPMS tour through the pickup (delivery) graph, given that the route of $z^{\mathrm{TSP}}_{\mathrm{D}}$ ($z^{\mathrm{TSP}}_{\mathrm{P}}$) is used as delivery (pickup) tour. $z_{\mathrm{P|D}}$ ($z_{\mathrm{P|D}}$) gives the corresponding upper bound for the DTSPMS obtained by $z^{\text{TSP-D}}_{\mathrm{P}} + z^{\mathrm{TSP}}_{\mathrm{D}}$ ($z^{\text{TSP-P}}_{\mathrm{D}} + z^{\mathrm{TSP}}_{\mathrm{P}}$), and $\frac{z_{\mathrm{P|D}}}{z^*}$ ($\frac{z_{\mathrm{D|P}}}{z^*}$) the gap between this upper bound and the optimal solution. $\frac{\bar{z}^{\mathrm{TSP}}}{z^*}$ shows the average gap between the upper bounds obtained from $z^{\text{TSP-D}}_{\mathrm{P}}$ and $z^{\text{TSP-P}}_{\mathrm{D}}$, and the optimal solution z^*. Finally, z^{TSP}_1 is the

upper bound obtained by solving the single stack problem (referred to as SS in Paper A), $\frac{z_1^{\text{TSP}}}{z^*}$ shows the ratio between the single stack upper bound and the optimal solution, and $\frac{z_1^{\text{TSP}}}{z^{\text{TSP}}}$ shows the ratio between the upper and lower bounds.

	$z_{\text{D}}^{\text{TSPD}}$	$z_{\text{P}}^{\text{TSPD}}$	$z_{\text{D}\|\text{P}}$	$z_{\text{P}\|\text{D}}$	$\frac{z_{\text{D}\|\text{P}}}{z^*}$	$\frac{z_{\text{P}\|\text{D}}}{z^*}$	z_1^{TSP}	$\frac{z_1^{\text{TSP}}}{z^*}$	$\frac{z_1^{\text{TSP}}}{z^{\text{TSP}}}$
R00	397	412	764	752	1.031	1.015	935	1.26	1.32
R01	376	443	754	785	1.000	1.041	971	1.29	1.35
R02	387	344	706	660	1.073	1.003	957	1.45	1.51
R03	416	440	793	796	1.033	1.036	977	1.27	1.33
R04	305	441	708	727	1.000	1.027	949	1.34	1.38
R05	369	425	740	770	1.004	1.045	980	1.33	1.37
R06	494	394	858	841	1.026	1.006	998	1.19	1.23
R07	360	405	724	692	1.049	1.003	916	1.33	1.41
R08	437	427	827	826	1.001	1.000	1048	1.27	1.33
R09	430	371	779	773	1.014	1.007	909	1.18	1.21
avg.	397	410	765	762	1.023	1.018	964	1.29	1.34

Table 2.2: Upper bounds, 15 orders.

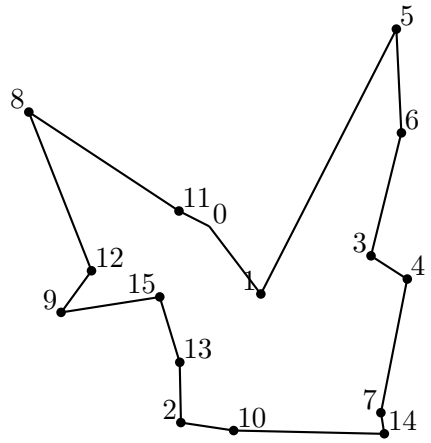
Table 2.2 shows that quite good DTSPMS solutions can be obtained by first solving a TSP on each of the graphs, and then solving the DTSPMS with this route fixed. By doing this for each of the routes (pickup and delivery) a solution can be obtained that is within 1% of the optimal in 8 of the 10 instances with 15 orders, as shown in the table. It is worth noting that instance R07, where the optimal DTSPMS solution consists of two routes that are around 5% from TSP-optimal, a solution exists that contains one TSP-optimal route and is only 0.3% above the optimal. As mentioned earlier, these results can be expected to deteriorate with increased problem size, but the trend is interesting, and indicates a path of research that could possibly be investigated further, however time has not allowed for this to be done at this point. The last two columns show that the single stack bound is rather weak, even for small instances.

Illustrations of the DTSPMS- and TSP-optimal solutions of two smaller instances are shown in Figures 2.6 and 2.7. These figures illustrate how one or both TSP-optimal tours may sometimes differ from the DTSPMS-optimal, and also how they can be very similar. Further illustrations of the DTSPMS-optimal and TSP-optimal solutions of Table 2.1 can be found in Appendix D.1.

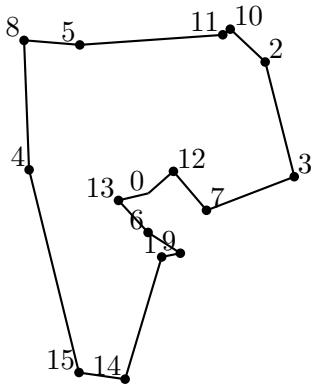
Larger instances

Some similar bounds can be calculated for the instances with 33 orders, as reported in Tables 2.3 and 2.4.

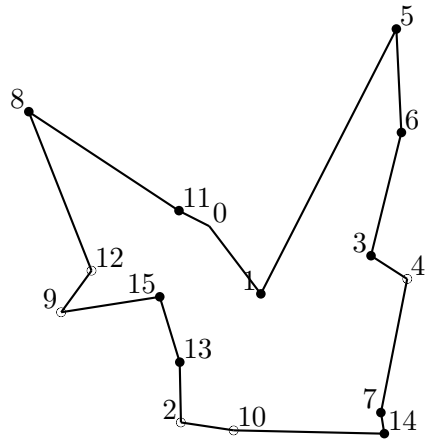
Table 2.3 is similar to Table 2.1, but reports values for the 33 order instances, using the best known solution to each of the 33 order problems



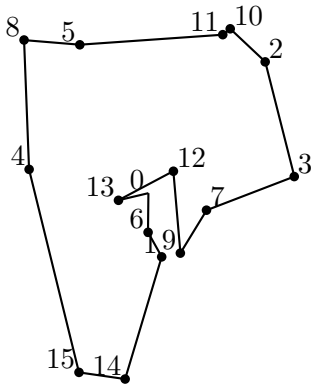
(a) TSP, pickup, 403.



(b) TSP, delivery, 286.



(c) DTSPMS, pickup, 403.

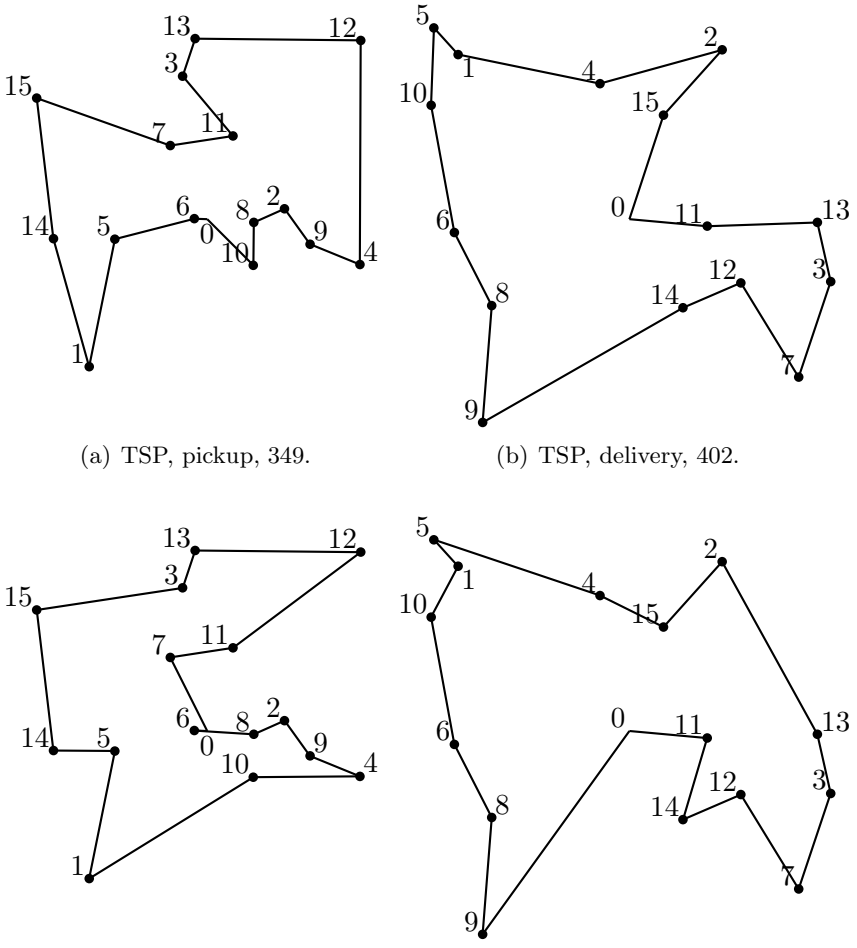


(d) DTSPMS, delivery, 305.

Figure 2.6: Comparison of optimal TSP ($z = 689$) and DTSPMS ($z = 708$) solutions for instance R04.

	z^{best}	$z_{\text{P}}^{\text{best}}$	$z_{\text{D}}^{\text{best}}$	z^{TSP}	$z_{\text{P}}^{\text{TSP}}$	$z_{\text{D}}^{\text{TSP}}$	$\frac{z^{\text{best}}}{z^{\text{TSP}}}$	$\frac{z_{\text{P}}^{\text{best}}}{z_{\text{P}}^{\text{TSP}}}$	$\frac{z_{\text{D}}^{\text{best}}}{z_{\text{D}}^{\text{TSP}}}$
R00	1063	528	535	911	482	429	1.17	1.10	1.25
R01	1032	551	481	875	471	404	1.18	1.17	1.19
R02	1065	563	502	935	504	431	1.14	1.12	1.16
R03	1100	515	585	961	494	467	1.14	1.04	1.25
R04	1052	550	502	937	511	426	1.12	1.08	1.18
R05	1008	515	493	900	479	421	1.12	1.08	1.17
R06	1110	512	598	998	457	541	1.11	1.12	1.11
R07	1105	603	502	963	481	482	1.15	1.25	1.04
R08	1109	580	529	978	492	486	1.13	1.18	1.09
R09	1091	517	574	976	464	512	1.12	1.11	1.12
avg.	1074	543	530	943	484	460	1.14	1.12	1.16

Table 2.3: Partial comparison of best known DTSPMS solution and TSP-optimal, 33 orders.



(a) TSP, pickup, 349.

(b) TSP, delivery, 402.

(c) DTSPMS, pickup, 359.

(d) DTSPMS, delivery, 409.

Figure 2.7: Comparison of optimal TSP ($z = 751$) and DTSPMS ($z = 768$) solutions for instance R09.

(from Table A.1), as the DTSPMS-optimal solutions are unknown for these instances. Again it can be observed that there is a significant variation in values of $\frac{z^{\text{best}}}{z^{\text{TSP}}}$ and $\frac{z_{\text{P}}^{\text{best}}}{z_{\text{P}}^{\text{TSP}}}$ for one instance – some instances form the best DTSPMS-solution by moderate modifications to each TSP-optimal route, while others perform only minor modifications (down to 4%) on one graph, and significant modifications to the other. Table 2.4

	z^{best}	z_1^{TSP}	$\frac{z_1^{\text{TSP}}}{z^{\text{best}}}$	$\frac{z_1^{\text{TSP}}}{z^{\text{TSP}}}$
R00	1063	1682	1.58	1.85
R01	1032	1579	1.53	1.80
R02	1065	1564	1.47	1.67
R03	1100	1741	1.58	1.81
R04	1052	1629	1.55	1.74
R05	1008	1438	1.43	1.60
R06	1110	1643	1.48	1.65
R07	1105	1696	1.53	1.76
R08	1109	1643	1.48	1.68
R09	1091	1556	1.43	1.59
avg.	1074	1617	1.51	1.72

Table 2.4: Comparison of upper bound, 33 orders.

shows the upper bounds that can be computed for 33 order instances $\frac{z_1^{\text{TSP}}}{z^{\text{TSP}}}$ (z_1^{TSP} are the numbers given as SS in Paper A). It has not been possible to compute the values $z_{\text{P|D}}$ and $z_{\text{D|P}}$ for these instances. The upper bound z_1^{TSP} is compared to the best known solutions and to the lower bound. Most notably the ratio between the upper and lower bounds has increased considerably with the increase of the problem size from 15 to 33 orders (from 1.34 to 1.72 on average), indicating that the strength of one or both bounds has also decreased significantly.

An interesting interpretation of the value $\frac{z^*}{z^{\text{TSP}}}$, is that it indicates the cost of not allowing repacking. If repacking were allowed the orders could be served at the value z^{TSP} instead of z^*, meaning that the cost increases by on average 4% by not allowing repacking for the 15 order instances, and by 14% for the 33 order instances.

2.3 Paper: Heuristic solution approaches to the DTSPMS

In the paper "The Double Travelling Salesman Problem with Multiple Stacks – Formulation and Heuristic Solution Approaches" (cf. Appendix A) a selection of metaheuristics for the DTSPMS are presented, implemented and tested. Additionally this paper constitutes the introduction of the DTSPMS to the research literature. This section will provide some additional comments as a supplement to the paper. It is recommended that the paper is read before reading the remainder of this section. It

should be noted that the symbols used in the paper differ slightly from the model presented in (2.11)–(2.22) and used throughout the rest of the thesis.

2.3.1 Additional solutions with known optimal solutions

By using the exact approaches that will be presented in Paper B, it has been possible to evaluate the heuristic solutions for instances with known optimal solutions, that are slightly larger than those presented in Section A.4.4. These instances have been tested using the large neighbourhood search algorithm, and the results are summarised in Table 2.5.

	opt.	10 sec.	180 sec.
R00-15	741	1.000 (3)	1.000 (3)
R01-15	754	1.003 (1)	1.000 (3)
R02-15	658	1.000 (3)	1.000 (3)
R03-15	768	1.002 (2)	1.000 (3)
R04-15	708	1.000 (3)	1.000 (3)
R05-15	737	1.000 (3)	1.000 (3)
R06-15	836	1.000 (3)	1.000 (3)
R07-15	690	1.000 (3)	1.000 (3)
R08-15	826	1.000 (3)	1.000 (3)
R09-15	768	1.001 (2)	1.000 (3)
R10-15	698	1.000 (3)	1.000 (3)
R11-15	684	1.000 (3)	1.000 (3)
R12-15	751	1.000 (3)	1.000 (3)
R13-15	744	1.000 (3)	1.000 (3)
R14-15	751	1.001 (1)	1.000 (3)
R15-15	748	1.000 (3)	1.000 (3)
R16-15	692	1.000 (3)	1.000 (3)
R17-15	783	1.000 (3)	1.000 (3)
R18-15	783	1.000 (3)	1.000 (3)
R19-15	800	1.000 (3)	1.000 (3)
avg.	746	< 1.001 ($\frac{54}{60}$)	1.000 ($\frac{60}{60}$)

Table 2.5: Larger instances with a known optimal solution.

As Table A.3, Table 2.5 shows the average deviation between the heuristic and the optimal solutions for three runs, and the number of times out of the three that the optimal solution is obtained. While the optimal solution was consistently found for the 12 order instances, even for the short running time, this is not quite the case for the 15 order instances. For 4 of the 20 tested instances the optimal solution is not always found, however the average deviation is still very small (within 0.3%). For the longer run time of 3 minutes the heuristic still consistently discovers the optimal solution.

2.3.2 General heuristic improvements

When implementing a solution method for a given problem, one always reaches a point where the refinement must be terminated, even though it is always possible to test new parameters and add new features. Of the methods presented in the paper, particularly the work on the tabu search method could be said to have been ended prematurely. The overall approach was to first implement a basic version of each of the algorithms, and then focus the further work on the algorithms that seemed most promising from an early stage. This meant that some of the heuristics presented in the paper are rather rudimentary.

Several common enhancements of the basic tabu search algorithm have already been mentioned briefly in the introductory section 1.3.1, such as variable tabu tenure. Additionally, the tabu search should probably have been allowed to choose a random operator at each iteration, rather than using a fixed pattern, and the argument of maintaining tabu search as a deterministic algorithm does not seem justified in hindsight. The termination criterion could have been changed to the more commonly used "a certain number of iterations without improvement", supplemented with a restart or some other diversification technique to make better use of the full time available. Furthermore, it would be tempting to test reducing the size of the explored neighbourhood in each iteration, by searching only part of the neighbourhood, for example by searching only until an improving solution has been found. Finally, each of the just mentioned suggestions could be made dependent on a parameter, so as to take more or less effect at the beginning/end of the calculation.

The implementation of iterated local search that was made for the paper was also rather crude. In particular each restart could have been performed from a solution that was more closely related to a previous good solution, rather than being restarted from a new random solution. As mentioned in the paper the initial idea was indeed to examine the behaviour of a simple steepest descent approach with restarts, but as this idea turned into an ILS algorithm, further known ILS features could have been tested, such as the implementation of a proper *kick* function to generate restart solutions. A potential kick function could be an extension of the Complete-Swap to swap 3 orders – this would be in line with the suggestion of Lourenço et al. [76] to use a higher order move of an existing neighbourhood as kick.

2.3.3 Initial solutions

The initial solutions used by the heuristics has been based on a heuristic solution to the single stack problem. An alternative initial solution could have been obtained by initially solving a TSP for one graph, and then implementing a simple construction heuristic to produce a good tour in the other graph based on a given loading. This would generate an initial

solution that could make better use of the multiple stacks. This approach would naturally lead to two different solutions (by solving either the pickup or the delivery route first) – in an implementation one might calculate both and use the best.

It is questionable whether the initial solution has a significant impact on the final solution, however as one TSP-optimal tour is part of the optimal DTSPMS-solution at least for the smaller instances (as shown in Table 2.1), and the best known solution for the larger instances often contains one tour that is much closer to its TSP-optimum than the other, such an approach might prove worthwhile.

2.3.4 Infeasible solutions

A common approach when implementing metaheuristics is to allow intermediate infeasible solutions by imposing an additional cost/penalty depending on the amount of infeasibility. This cost is then added to the objective function when expressing the value of a solution, and the feasibility of a solution must then be considered whenever the incumbent solution is updated. The idea of allowing infeasible solutions is already mentioned in the paper as a suggestion for further research.

Each of the implemented metaheuristics could be expanded to allow intermediate infeasible solutions. To this end a meaningful measure of infeasibility would need to be determined. Traditionally, when implementing solution approaches that allow intermediate infeasibilities, such infeasibilities can often be expressed in terms of numeric violation of a resource constraints (e.g. of the type $\sum x \leq \bar{x}$; excessive usage of some resource, such as loading capacity or time windows), and the violation can thus be quantified by determining the numeric size of the violation ($\sum x - \bar{x}$).

Infeasible solutions for the DTSPMS can be constructed in two different ways: by allowing loading plans that are infeasible regarding the ordering of the items, or by allowing loading plans where some rows may exceed the row capacity. In the latter case the degree of infeasibility can quite easily be expressed in terms of the amount of excess row capacity that is used. For the former case such an approach is not immediately feasible, however this approach can be seen as a violation of the "no intermediate repacking" constraint, and thus the degree of infeasibility could probably be expressed in terms of "number of intermediate rearrangements required to perform the delivery tour".

2.3.5 Alternative operators

When developing the *route-swap* and *complete-swap* operators, focus was on constructing operators that maintained the feasibility of the solution. This meant that the use of very simple operators was ruled out from the

beginning. Furthermore it did not seem possible to construct a single operator that maintained feasibility and would still be able to cover the entire solution space.

When allowing infeasible intermediate solutions, several simpler operators immediately become available, including the traditional ones used for the TSP/VRP.

An approach that allows infeasible solutions and thereby the usage of simpler operators has in fact been succesfully tested by Felipe et al. [34], which will be further discussed in 2.3.9 (p. 53).

As mentioned earlier most operators developed for the TSPPD/TSP-PDL focus on the preservation of the LIFO constraints, and are hence not directly applicable to the DTSPMS. In particular the idea of moving entire *blocks* can not be immediately transferred, as blocks of consecutive items in a loading row does not correspond to a contiguous routing unit in the DTSPMS.

The block concept can be used to illustrate the route-swap introduced in the paper. If the pickup visits c^+ and d^+ of Figure 2.4 (p. 29) are swapped in the pickup route without updating the delivery route, the resulting infeasibility is clearly visible from the cross seen in Figure 2.8.

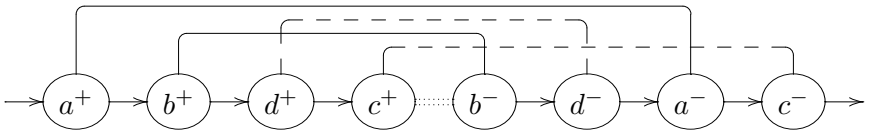


Figure 2.8: Blocks used to illustrate an infeasibility in the DTSPMS.

It is also easily seen from Figure 2.4 why no further modifications are necessary when swapping two visits that are assigned to different rows, e.g. swapping b^+ and c^+ would lead to Figure 2.9, which contains no crosses.

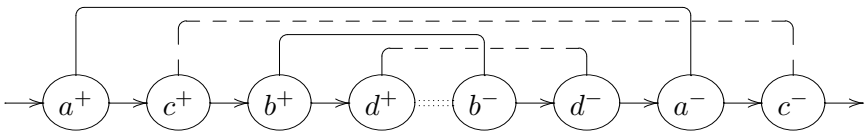


Figure 2.9: Illustration of blocks in the DTSPMS.

In fact this leads to the idea of allowing shifts of not only one, but several positions, since the swaps can be performed without affecting the row assignment or the other route, as long as the swapped orders are assigned to different rows.

The complete-swap performs a swap of both pickup and delivery order of two requests, and thus in an illustration such as Figure 2.9 only the labels would change when performing a complete-swap (for example the labels of orders a and d could be swapped, while the lines above, indicating loading compatibility, would remain unchanged).

Unfortunately, it has not been possible thus far to convert the concept of blocks to a form that is immediately useful for constructing new neighbourhoods for the DTSPMS, by modification of existing TSP neighbourhoods.

For the large neighbourhood search algorithm a random removal operator could have been implemented, to supplement the relatedness and price-based ones and provide further diversification.

2.3.6 Switching strategies

The method of combining the two neighbourhood structures could have been modified, to develop adaptively over the course of the iterations. As mentioned earlier the reason for using a combination of operators was initially a necessity to properly examine the entire solution space, but the similarities with VNS could be further exploited by using a more advanced switching strategy, such as the one from VNS where one operator at a time is applied, and more involved operators are only applied whenever the simpler operators has been unable to produce improving solutions for a given period.

Alternatively, an adaptive probability could be used, such that the probability of choosing a given operator at each iteration would depend on the historic performance of this operator over recent iterations. This idea is similar to the adaptive large neighbourhood search presented in Røpke and Pisinger [96] and Pisinger and Røpke [93].

2.3.7 Termination criterion

As described in the paper, running time was chosen as termination criterion, rather than e.g. number of iterations, since it better matched the requirements of the real-world situation where the problem would occur. It also eased the comparison of the different heuristics, by reducing the number of performance parameters to compare. Through the parameter tuning process each algorithm should be allowed to perform at its best, even with a pre-set running time.

To ensure reproducibility and stability of the results, the use of running time as termination criterion was obtained by converting this to a number of iterations. This was done by first determining the number of iterations that could be completed during the assigned running time as an average over three timed runs, and then completing the tests using the resulting number of iterations to represent the desired running time. This was done for 180 seconds, and all other running times used were found by suitable multiplication. To further reduce the impact of time variations each timed run was repeated three times, and the final number of iterations was taken as an average over these three.

The number of iterations to be completed within a given time span was

dependent on some of the parameters for each algorithm, namely:

ILS: percentage of moves of each type
TS: percentage of moves of each type
SA: start and end temperature
LNS: number of orders to remove at each iteration (min and max)

Thus the number of iterations was determined for each combination of these parameters, and these numbers were used for calibration runs and for the final runs.

Furthermore each of the final runs were repeated three times with different random seeding for all algorithms that include randomness (all but tabu search), to reduce the impact of this, and give a broader view of the results.

2.3.8 SA temperature reduction scheme

The temperature reduction scheme implemented for simulated annealing to allow the use of running time as termination criterion is described rather briefly in the paper (Section A.3.5), and will be described in further detail here.

Traditionally the temperature reduction function used by simulated annealing is based on the reduction scheme

$$T_{i+1} = c \cdot T_i \tag{2.23}$$

where $T_0 = T_s$, $c \in (0; 1)$ is some reduction factor, and the algorithm terminates when the final temperature T_e is reached, that is the temperature is reduced exponentially as a function of the number of iterations completed, as seen in Figure 2.10. The purpose of this modification corresponds to determining a value of c that ensures that the desired ending temperature is reached at the desired ending time of the algorithm.

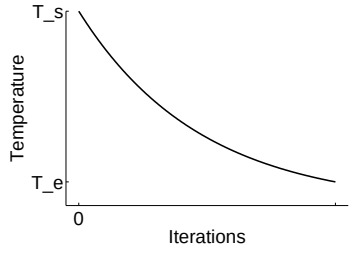


Figure 2.10: Standard cooling scheme.

The temperature reduction function given in (2.23) is equivalent to finding the temperature $T(i)$ at iteration i as:

$$T(i) = T_s \cdot c^i \tag{2.24}$$

where T_s is again the starting temperature, c is the reduction factor and the algorithm is stopped once the desired final temperature T_e is reached.

If instead a certain number of iterations is allowed, the temperature reduction function can be expressed as

$$T(i) = T_s \cdot \left(\frac{T_e}{T_s}\right)^{\frac{i}{i_{\max}}} \tag{2.25}$$

where $T(i)$ is the temperature at iteration i, and $i_{\max}$ is the predetermined maximum allowed number of iterations. This ensures that $T(0) = T_s$, $T(i_{\max}) = T_e$ and the behaviour between these two points corresponds to that of (2.23) when temperature T_e is reached at time $t_{\max}$.

Similarly the temperature can be calculated as a function of the running time

$$T(t) = T_s \cdot \left(\frac{T_e}{T_s}\right)^{\frac{t}{t_{\max}}} \tag{2.26}$$

where $t_{\max}$ is the total allowed running time.

For the implementation in the paper this approach has been used to allow the algorithm a predetermined running time, and still ensure that an appropriate temperature interval is covered, effectively keeping the same exponential temperature reduction curve with changed x-axis, as illustrated by Figure 2.11.

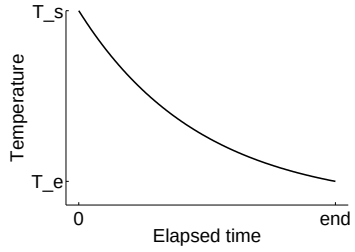


Figure 2.11: Adjusted cooling scheme.

2.3.9 Related work by others

The work presented in A has inspired Felipe et al. [34, 35] to work on the DTSPMS, implementing a variable neighbourhood search algorithm. This implementation uses a combination of the feasible operators introduced in the paper of Appendix A and new operators that are in part based on the existing ones. Their approach partially allows intermediate solutions that are capacity infeasible.

One new operator that is introduced (*in-stack swap*) is similar on the complete-swap, but performs the swap on two orders that are assigned to the same loading row, thus affecting only the routes, and not the

row assignment. A *reinsertion* operator is introduced, which removes an order from the solution and reinserts it in a different row. In order to work well the Reinsertion operator requires that infeasible solutions are allowed (exceeding row capacity), and it is used in combination with a repair heuristic, which attempts to restore feasibility of a solution. An *r-route permutation* operator is introduced, which permutes a sequence of customers in one route. The sequence is constituted of consecutive customers assigned to different rows, such that one route can be modified without affecting the other elements of the solution. Figure 2.12 points out some sequences of customers that can be permuted this way. When 2 customers are permuted this operator corresponds to the route-swap (specifically the case where the swapped customers are assigned to different rows). Finally, a *stack permutation* operator is a generalisation of the complete-swap, allowing more than two orders to be permuted in one step, and a variation called *r-complete stack permutation* performs multiple simultaneous stack permutations on different parts of the row.

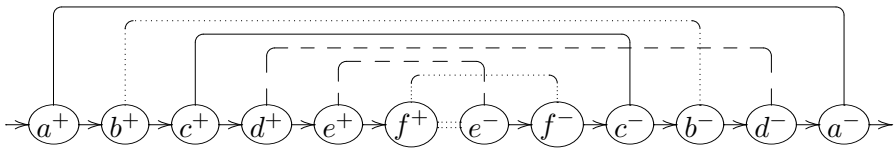


Figure 2.12: Illustration of the Route Permutation, examples of permutable sequences are (b^+, c^+, d^+), (e^-, f^-, c^-), (c^-, b^-, d^-), (b^-, d^-, a^-).

Three variations of the VNS are implemented, Variable Neighbourhood Descent, which allows no hill-climbing moves, a Generalised Variable Neighbourhood Search, which uses a nested VND as the local search procedure and allows ways to escape local optima, and a Hybridised VNS, which is based on the GVNS, adding further features, such as restarts, tabu lists, and randomised operator choice.

The results presented by Felipe et al. [35] are generally better than those of paper A for all but the smallest instances (12 customers), and for a third of the largest instances (66 customers) an improved best solution is presented. Finally some instances with 132 customers are generated and tested for a 3-row setup, although instances with a row capacity of 44 does not seem very likely in real-life.

Felipe et al. [35] also compare the impact of each of the operators used. Interestingly the by far best operators are the reinsertion, which uses infeasible solutions, and the complete-swap. Particularly, a comparison of the complete-swap and the 2 new operators based on a related idea (in-stack swap and complete stack permutation) show quite different performance.

The findings of Felipe et al. [35] support the suspicions from Paper A, that the DTSPMS is more easily solved when the algorithm can move rather freely through the solution space, either by applying far-reaching

operators, or by allowing intermediate infeasible solutions.

2.4 Paper: Exact solutions to the DTSPMS

In addition to the models presented in Paper B, two further modelling approaches were developed for the DTSPMS, however the results were not promising, and they were excluded from the published work. Nonetheless, they are presented here, to give an impression of the variety of modelling approaches that can be applied to the DTSPMS. These models represent supplementary work to the paper, and it is thus recommended to read the paper before reading the remainder of this section.

2.4.1 Variation of the flow model: The multiple commodities flow model

This section considers a variation of the flow model presented in Section B.4, called the *multiple commodities flow model*. The DTSPMS can be modelled in terms of multi-commodity flows, with one commodity for each available loading row, and a supply/demand of 1 for each customer. Since the assignment of orders to rows is a decision variable, so is the commodity that is supplied/demanded by each customer.

The idea is to maintain the LIFO constraints by comparing the load of the vehicle on pickup and delivery of each item and is inspired by model TSPPDL2 (see Cordeau et al. [23]). If node i supplies/demands commodity r, then the vehicle load of this commodity must be the same immediately before order i is picked up as it is immediately after i has been delivered. In Cordeau et al. [23] a new set of variables must be added to the model when the LIFO constraints are formulated this way, whereas in the DTSPMS this set of variables simply replaces the y variables from the precedence model of Section B.3.1, and thus the disadvantage is potentially smaller. The model requires some big M notation, however, the value of M is equal to the row length, and typically not much greater than 10 in practical applications.

The *multiple commodities flow model* uses three sets of variables, two of which are binary:

$$x_{ij}^T = \begin{cases} 1 & \text{if arc } (i,j) \text{ is used in graph } G^T, \\ 0 & \text{otherwise,} \end{cases}$$

$$z_i^r = \begin{cases} 1 & \text{if item } i \text{ is placed in row } r, \\ 0 & \text{otherwise,} \end{cases}$$

u_{ij}^{Tr} is the load of commodity r carried on arc (i,j).

The x and z variables are the same as in the precedence formulation, while the u variables have changed from indicating the (binary) prece-

dence between any pair of nodes to indicating the (integer) ordering of the nodes for each row. Since all nodes have a supply or demand of 1, the load of the vehicle when leaving a node corresponds to the position of that node in the row.

This use of the u variables resembles the idea introduced by Miller, Tucker and Zemlin in 1960 (Miller et al. [78]) for subtour elimination, where integer variables were used to indicate the order of the visits.

The multiple commodities flow model can be expressed as follows:

$$\min \sum_{\substack{T\in\mathcal{T}\\ i,j\in V^T}} c_{ij}^T \cdot x_{ij}^T \tag{2.27}$$

subject to

$$\sum_{j\in V_C^T} u_{ij}^{\mathrm{P}r} = \sum_{j\in V_C^T} u_{ji}^{\mathrm{P}r} + z_i^r \quad i \in V_C^T, r \in \mathcal{R} \tag{2.28}$$

$$\sum_{j\in V_C^T} u_{ij}^{\mathrm{D}r} = \sum_{j\in V_C^T} u_{ji}^{\mathrm{D}r} - z_i^r \quad r \in \mathcal{R}, i \in V_C^T \tag{2.29}$$

$$\sum_{j\in V_C^T} u_{0j}^{\mathrm{P}r} = \sum_{j\in V_C^T} u_{j0}^{\mathrm{D}r} = 0 \quad r \in \mathcal{R} \tag{2.30}$$

$$\sum_{i\in V_C^T} u_{i0}^{\mathrm{P}r} = \sum_{i\in V_C^T} u_{0i}^{\mathrm{D}r} \quad r \in \mathcal{R} \tag{2.31}$$

$$\sum_{r\in\mathcal{R}} z_i^r = 1 \quad i \in V_C^T \tag{2.32}$$

$$\sum_{i\in V_C^T} z_i^r \le L \quad r \in \mathcal{R} \tag{2.33}$$

$$z_0^r = 0 \quad r \in \mathcal{R} \tag{2.34}$$

$$\sum_{j\in V^T} x_{ji}^T = \sum_{j\in V^T} x_{ij}^T = 1 \quad T \in \mathcal{T}, i \in V^T \tag{2.35}$$

$$u_{ij}^{Tr} \le L \cdot x_{ij}^T \quad T \in \mathcal{T}, r \in \mathcal{R}, i, j \in V_C^T \tag{2.36}$$

$$\sum_{j\in V_C^T} u_{ij}^{\mathrm{P}r} - \sum_{j\in V_C^T} u_{ji}^{\mathrm{D}r} \le L(1 - z_i^r) \quad r \in \mathcal{R}, i \in V_C^T \tag{2.37}$$

$$\sum_{j\in V_C^T} u_{ij}^{\mathrm{P}r} - \sum_{j\in V_C^T} u_{ji}^{\mathrm{D}r} \ge -L(1 - z_i^r) \quad r \in \mathcal{R}, i \in V_C^T \tag{2.38}$$

$$x_{ij}^T \in \mathbb{B} \quad T \in \mathcal{T}, i, j \in V^T, i \ne j \tag{2.39}$$

$$u_{ij}^{Tr} \in \mathbb{R} \quad T \in \mathcal{T}, r \in \mathcal{R}, i, j \in V_C^T, i \ne j \tag{2.40}$$

$$z_i^r \in \mathbb{B} \quad r \in \mathcal{R}, i \in \mathcal{V}_C^{\mathrm{P}} \cup \mathcal{V}_C^{\mathrm{D}}. \tag{2.41}$$

Constraints (2.28) and (2.29) ensure flow conservation in the pickup and delivery graphs, respectively. The supply of commodity r is z_i^r for each

node i in the pickup graph, and similarly the demand is z_i^r for each node i in the delivery graph. Constraints (2.30) ensure that the vehicle is empty at the start and end points, while (2.31) ensure that the loads are maintained before and after the intermediate long haul. (2.31) is superfluous when $N = R \times L$, since all rows will then be full. Constraints (2.32) guarantee that all customers are assigned a supply/demand of exactly one commodity, and (2.33) enforce row capacity, i.e. that only a limited number of customers are assigned commodity r. (2.33) hold with equality when $N = R \times L$. The depot is not assigned to any row through (2.34) while constraints (2.35) ensure that all customers are visited. Constraints (2.36) ensure that no load is carried on unused arcs. This must be expressed explicitly since there is no cost for carrying load. Finally, constraints (2.37)–(2.38) ensure the LIFO constraints; if $z_i^r = 1$, then $\sum_j u_{ij}^{\mathrm{P}r} = \sum_j u_{ji}^{\mathrm{D}r}$ must hold, otherwise no binding exists.

2.4.2 Expanded model: 4 variable formulation

Additionally, attempts were made at solving an expanded model where the w-variables from B.3.2 were added to the original precedence model from (B.2)–(B.12), as briefly mentioned in the paper. The hope was that by combining the different sets of variables, and adding the possible constraints between them, it would be possible to provide closer ties between the variables, hence counteracting the effects of the matrix structure illustrated in Figure B.2.

The complete variable set thus becomes:

$$x_{ij}^T = \begin{cases} 1 & \text{if arc } (i,j) \text{ is used in graph } T, \\ 0 & \text{otherwise,} \end{cases}$$

$$y_{ij}^T = \begin{cases} 1 & \text{if } v_i^T \text{ is visited before } v_j^T, \\ 0 & \text{otherwise,} \end{cases}$$

$$w_{ij}^r = \begin{cases} 1 & \text{if item } i \text{ is picked up before item } j \text{ and both are in row } r, \\ 0 & \text{otherwise,} \end{cases}$$

$$z_i^r = \begin{cases} 1 & \text{if item } i \text{ is placed in row } r, \\ 0 & \text{otherwise.} \end{cases}$$

The full set of constraints that were implemented for the 4 variable model is listed in (2.42)–(2.69).

$$\sum_{i \in V^T} x_{ij}^T = 1 \qquad j \in V^T \tag{2.42}$$

$$\sum_{j \in V^T} x_{ij}^T = 1 \qquad i \in V^T \tag{2.43}$$

$$y_{ij}^T + y_{ji}^T = 1 \qquad T \in \mathcal{T}, i, j \in V_C^T \tag{2.44}$$

$$y_{ik}^T + y_{kj}^T \leq y_{ij}^T + 1 \qquad T \in \mathcal{T} i, j, k \in V_C^T \tag{2.45}$$

$$x_{ij}^T \leq y_{ij}^T \qquad T \in \mathcal{T}, i, j \in V_C^T \tag{2.46}$$

$$y_{ij}^{\mathrm{P}} + z_i^r + z_j^r \leq 3 - y_{ij}^{\mathrm{D}} \qquad r \in \mathcal{R} i, j \in V_C^T \tag{2.47}$$

$$\sum_{r \in \mathcal{R}} z_i^r = 1 \qquad i \in V_C^T \tag{2.48}$$

$$\sum_{i \in \mathcal{V}_C^{\mathrm{P}} \cup \mathcal{V}_C^{\mathrm{D}}} z_i^r \leq L \qquad r \in \mathcal{R} \tag{2.49}$$

$$w_{ij}^r + w_{ji}^r \geq z_i^r + z_j^r - 1 \qquad r \in \mathcal{R}, i, j \in V_C^T \tag{2.50}$$

$$2 \cdot (w_{ij}^r + w_{ji}^r) \leq z_i^r + z_j^r \qquad r \in \mathcal{R}, i, j \in V_C^T \tag{2.51}$$

$$\sum_{i,j \in V_C^T} y_{ij}^T = \frac{n \cdot (n-1)}{2} \qquad T \in \mathcal{T} \tag{2.52}$$

$$\sum_{j \in V_C^T} y_{ij}^T + \sum_{j \in V_C^T} y_{ji}^T = n - 1 \qquad T \in \mathcal{T}, i \in V_C^T \tag{2.53}$$

$$\sum_{i,j \in V_C^T} w_{ij}^r = \frac{L \cdot (L-1)}{2} \qquad r \in \mathcal{R} \tag{2.54}$$

$$\sum_{j \in V_C^T} w_{ij}^r + \sum_{j \in V_C^T} w_{ji}^r = (L-1) \cdot z_i^r \qquad r \in \mathcal{R}, i \in V_C^T \tag{2.55}$$

$$w_{ij}^r \leq y_{ij}^{\mathrm{P}} \qquad r \in \mathcal{R}, i, j \in V_C^T \tag{2.56}$$

$$w_{ij}^r + w_{ji}^r \leq 1 \qquad r \in \mathcal{R}, i, j \in V_C^T \tag{2.57}$$

$$\sum_{r \in \mathcal{R}} w_{ij}^r \leq 1 \qquad i, j \in V_C^T \tag{2.58}$$

$$\sum_{i \in V_C^T} w_{ij}^r \leq L \cdot z_j^r \qquad r \in \mathcal{R}, j \in V_C^T \tag{2.59}$$

$$\sum_{j \in V_C^T} w_{ij}^r \leq L \cdot z_i^r \qquad r \in \mathcal{R}, i \in V_C^T \tag{2.60}$$

$$z_i^r + z_j^r + y_{ij}^{\mathrm{P}} + y_{ji}^{\mathrm{D}} \geq 4 \cdot w_{ij}^r \qquad r \in \mathcal{R}, i, j \in V_C^T \tag{2.61}$$

$$z_i^r + z_j^r + y_{ij}^{\mathrm{P}} \leq 2 + w_{ij}^r \qquad r \in \mathcal{R}, i, j \in V_C^T \tag{2.62}$$

$$z_i^r + z_j^r + y_{ij}^{\mathrm{D}} \leq 2 + w_{ji}^r \qquad r \in \mathcal{R}, i, j \in V_C^T \tag{2.63}$$

$$x_{ij}^{\mathrm{P}} + w_{ji}^r \leq 1 + w_{ij}^r \qquad r \in \mathcal{R}, i, j \in V_C^T \tag{2.64}$$

$$x_{ij}^{\mathrm{D}} + w_{ij}^r \leq 1 + w_{ji}^r \qquad r \in \mathcal{R}, i, j \in V_C^T \tag{2.65}$$

$$x_{ij}^T \in \mathbb{B} \qquad T \in \mathcal{T}, i, j \in V^T, i \neq j \tag{2.66}$$

$$y_{ij}^T \in \mathbb{B} \qquad T \in \mathcal{T}, i, j \in V_C^T, i \neq j \tag{2.67}$$

$$w_{ij}^r \in \mathbb{B} \qquad r \in \mathcal{R}, i, j \in V_C^T, i \neq j \tag{2.68}$$

$$z_i^r \in \mathbb{B} \qquad r \in \mathcal{R}, i \in V_C^T. \tag{2.69}$$

Constraints (2.42)–(2.49) are identical to (2.12)–(2.19). The remaining constraints express various valid relations between the variables, but will not be explained individually in detail.

Constraints (2.50)–(2.51) express the connection between the z and w variables, and (2.52)–(2.53) use the knowledge of the total number of y variables that must be set to 1. (2.54)–(2.55) use similar knowledge of the w variables, and are only valid for instances with $n = R \cdot L$. (2.56)–(2.60) express further bounds on the w variables, while (2.61)–(2.65) state connections between w and the other variables. In particular (2.64) express that $x^{\mathrm{P}}_{ij} = 1 \Rightarrow w^r_{ij} \geq w^r_{ji}$, and similarly (2.65).

2.5 Extensions

So far, initial work on the DTSPMS has been presented, and the problem has been introduced to the research community. Further work on the DTSPMS can now be carried out, either by application of new or improved solution methods to the DTSPMS, or by extension of the problem into new contexts. This section will discuss a variety of such possible extensions, and comment on their implications.

The DTSPMS can naturally be generalised in several directions, by extension towards existing and related fields of research, or towards more realistic real-life applications. Such extensions include the DVRPMS (double VRP with multiple stacks) on which some initial work will be presented in the following. Later in this section some further possible directions of extension will be discussed, both in terms of integration with existing related problems from the literature, and more generally, possible extensions of the problem into other real-life scenarios. An example of the former would be a merger between the DTSPMS and the SVRPPDL, leading to a pickup and delivery problem with multiple loading stacks and mixed pickup and delivery operations. An example of the latter could be obtained by adding further complications that typically occur in real-life situations, such as time windows, a heterogenous fleet, and compatibility requirements between orders, or between orders and vehicles.

2.5.1 A generalisation of the DTSPMS: The DVRPMS

The most immediate extension of the DTSPMS seems to be the *DVRPMS* (Double Vehicle Routing Problem with Multiple Stacks). The Double VRP with Multiple Stacks is an extension of the DTSPMS to include multiple vehicles, hence the change of TSP to VRP in the name, yet still maintaining the availability of several LIFO loading rows in each vehicle. Here, a mathematical formulation of the DVRPMS will be given, based on the formulation presented for the DTSPMS, and some comments will be given on how the DVRPMS could be treated, in the light of the DTSPMS solution methods that have already been presented.

The DVRPMS would imply an extension of the problem to contain more than one container, and thus the construction of more than one route

in each of the graphs. The assignment of orders to containers/vehicles would then naturally become a decision variable, in addition to the row assignment variables already present in the DTSPMS.

An extension of the mathematical model (2.11)–(2.22) to include multiple vehicles is relatively straightforward, and can be obtained by adding a vehicle index $h \in \mathcal{H}$ to all 3 sets of variables, x_{ij}^{Th}, y_{ij}^{Th} and z_i^{rh}, and updating their definitions as follows:

$$x_{ij}^{Th} = \begin{cases} 1 & \text{if arc } (i,j) \text{ is travelled in graph } G^T \text{ by vehicle } h, \\ 0 & \text{otherwise,} \end{cases}$$

$$y_{ij}^{Th} = \begin{cases} 1 & \text{if node } v_i^T \text{ is visited before node } v_j^T \ (i \neq j), \\ & \quad \text{and both are visited by vehicle } h, \\ 0 & \text{otherwise,} \end{cases}$$

$$z_i^{rh} = \begin{cases} 1 & \text{if item } i \text{ is placed in row } r \text{ of vehicle } h, \\ 0 & \text{otherwise.} \end{cases}$$

The size of the model can be expected to increase considerably with this extension, since most constraints are repeated for every vehicle.

For most constraints the extension is simple, however constraints (2.14) which ensure that the precedence is defined for any pair of customers, must now only be imposed on any pair of customers which is visited by one and the same vehicle:

$$y_{ij}^{Th} + y_{ji}^{Th} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are both serviced by vehicle } h, \\ 0 & \text{otherwise.} \end{cases} \tag{2.70}$$

This can be expressed in a non-linear way, as:

$$y_{ij}^{Th} + y_{ji}^{Th} = \sum_{r \in \mathcal{R}} z_i^{rh} \cdot \sum_{r \in \mathcal{R}} z_j^{rh} \qquad T \in \mathcal{T}, h \in \mathcal{H}, i,j \in \mathcal{V}_C^T \tag{2.71}$$

since the precedence variables must be set exactly when two items are transported by the same vehicle, but not for items that are assigned to different vehicles. Constraints 2.71 can be linearised as follows:

$$y_{ij}^{Th} + y_{ji}^{Th} \geq \sum_{r \in \mathcal{R}} z_i^{rh} + \sum_{r \in \mathcal{R}} z_j^{rh} - 1 \qquad T \in \mathcal{T}, h \in \mathcal{H}, i,j \in \mathcal{V}_C^T \tag{2.72}$$

$$y_{ij}^{Th} + y_{ji}^{Th} \leq \frac{\sum_{r \in \mathcal{R}} z_i^{rh} + \sum_{r \in \mathcal{R}} z_j^{rh}}{2} \quad T \in \mathcal{T}, h \in \mathcal{H}, i,j \in \mathcal{V}_C^T. \tag{2.73}$$

To summarise, the DVRPMS can be formulated as:

$$\sum_{\substack{h\in\mathcal{H},T\in\mathcal{T}\\ i,j\in\mathcal{V}^T}} c_{ij}^{Th} x_{ij}^{Th} \tag{2.74}$$

$$\text{s.t.} \quad \sum_{h\in\mathcal{H},i\in\mathcal{V}^T} x_{ij}^{Th} = 1 \qquad T\in\mathcal{T}, j\in\mathcal{V}_C^T \tag{2.75}$$

$$\sum_{h\in\mathcal{H},i\in\mathcal{V}^T} x_{i0}^{Th} = |\mathcal{H}| \qquad T\in\mathcal{T} \tag{2.76}$$

$$\sum_{i\in\mathcal{V}^T} x_{ij}^{Th} = \sum_{i\in\mathcal{V}^T} x_{ji}^{Th} \qquad T\in\mathcal{T}, h\in\mathcal{H}, j\in\mathcal{V}_C^T \tag{2.77}$$

$$\sum_{r\in\mathcal{R}} z_i^{rh} \cdot \sum_{r\in\mathcal{R}} z_j^{rh} = y_{ij}^{Th} + y_{ji}^{Th} \qquad T\in\mathcal{T}, h\in\mathcal{H}, i,j\in\mathcal{V}_C^T \tag{2.78}$$

$$x_{ij}^{Th} \le y_{ij}^{Th} \qquad T\in\mathcal{T}, h\in\mathcal{H}, i,j\in\mathcal{V}_C^T \tag{2.79}$$

$$y_{ik}^{Th} + y_{kj}^{Th} \le y_{ij}^{Th} + 1 \qquad T\in\mathcal{T}, h\in\mathcal{H}, i,j,k\in\mathcal{V}_C^T \tag{2.80}$$

$$y_{ij}^{\mathrm{P}h} + z_i^{rh} + z_j^{rh} \le 3 - y_{ij}^{\mathrm{D}h} \qquad h\in\mathcal{H}, r\in\mathcal{R}, i,j\in\mathcal{V}_C^T \tag{2.81}$$

$$\sum_{r\in\mathcal{R},h\in\mathcal{H}} z_i^{rh} = 1 \qquad i\in\mathcal{V}_C^T \tag{2.82}$$

$$\sum_{i\in\mathcal{V}_C^{\mathrm{P}}\cup\mathcal{V}_C^{\mathrm{D}}} z_i^{rh} = L \qquad h\in\mathcal{H}, r\in\mathcal{R} \tag{2.83}$$

$$x_{ij}^{Th} \in \mathbb{B} \qquad T\in\mathcal{T}, h\in\mathcal{H}, i,j\in\mathcal{V}^T; i\neq j \tag{2.84}$$

$$y_{ij}^{Th} \in \mathbb{B} \qquad T\in\mathcal{T}, h\in\mathcal{H}, i,j\in\mathcal{V}_C^T; i\neq j \tag{2.85}$$

$$z_i^{rh} \in \mathbb{B} \qquad h\in\mathcal{H}, r\in\mathcal{R}, i\in\mathcal{V}_C^{\mathrm{P}}\cup\mathcal{V}_C^{\mathrm{D}}. \tag{2.86}$$

The objective function (2.74) again expresses the sum of the costs of all travelled arcs of the graph. (2.75) ensure that each customer node is visited, (2.76) that all vehicles are used, and (2.77) ensure flow balance. (2.78) are the constraints ensuring the precedence relation between any pair of customers serviced by the same vehicle, and can be linearised as shown in (2.72)–(2.73). Constraints (2.79)–(2.83) are similar to the original model for DTSPMS, and express that each travelled arc enforces a precedence, transitivity of the precedence variables, the connection between pickup and delivery route for each row, that each order is assigned to a row, and the row capacity.

In accordance with the DTSPMS it is assumed that the full capacity of all vehicles is used, thus (2.76) is stated using *all* vehicles, rather than using *at most* all vehicles.

The fact that each pickup/delivery pair must be visited by the same vehicle is ensured by (2.75), (2.78), and (2.79) in combination. (2.75) ensure that each node is visited, (2.78) that some y-variable will be set

corresponding to that visit, and (2.79) that the y-variables can only be set for a pair of nodes that are visited by the same vehicle.

2.5.1.1 Solving the DVRPMS

To solve the DVRPMS, a large part of the already presented heuristics could be reused, in one of two possible ways: Either by extending the neighbourhood structure to accomodate swaps/transfers between vehicles, or by using the DTSPMS as a subroutine that is applied for each vehicle.

The previously presented heuristic could be used as a subroutine for a separate routine dedicated to dividing the customers between vehicles. This would depend on the speed of the original heuristics, in order to be a useful approach. In short time it should be possible to produce solutions that are sufficiently good, or that at least compare identically (given two clusterings A and B, if A gives the better routing cost after few iterations, this should indicate that A is also better in terms of the optimal solutions to the routing problems). An indication of whether this is the case can be obtained by plotting the solution value improvement for the DTSPMS as a function of the iteration count/elapsed time, for a number of different instances. Such a plot can be seen in Figure 2.13, which shows the development of the solutions of 10 different instances, during 500 iterations (approximately 6 seconds), with the values of the best known solutions indicated on the right-most side. The figure shows

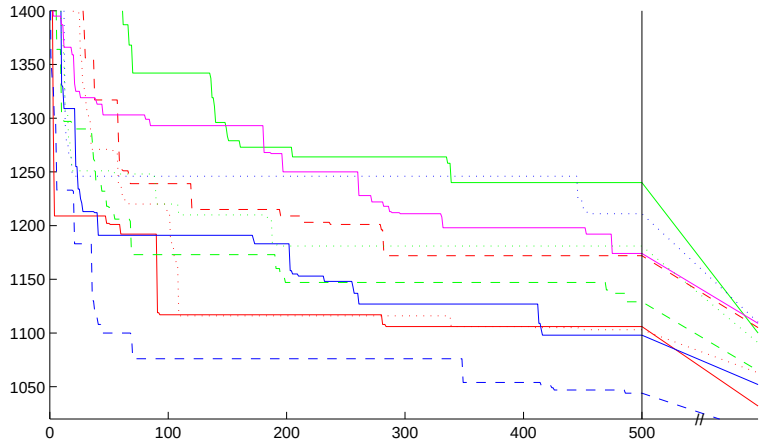


Figure 2.13: Development of solution values over time for different problem instances. Up to 500 iterations the progress of the algorithm is shown, after 500 iterations the best known objective value for each instance is plotted. Large Neighbourhood Search using the standard setting for short runs, with 3-15 orders removed per iteration.

that there are still noticeable improvements occuring up to iteration 500,

and there is considerable variation in how the solution after 500 iterations compares to the best known solutions. In combination with the running time of 6 seconds for the 500 iterations, this indicates that the current best solution heuristic for the DTSPMS may not be fast enough to be used as a subroutine in solving the DVRPMS. A similar test has been made with changed settings, to allow a higher number of iterations to be completed in shorter time. Figure 2.14 shows the results for 100 iterations

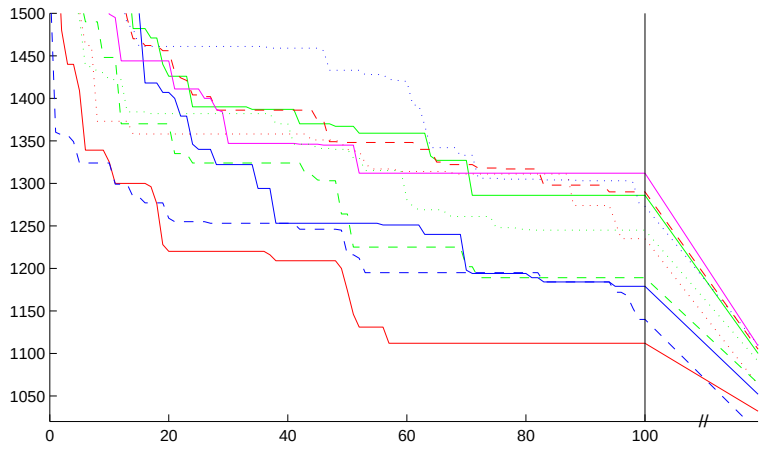


Figure 2.14: Development of solution values over time for different problem instances. Up to 100 iterations the progress of the algorithm is shown, after 100 iterations the best known objective value for each instance is plotted. Large Neighbourhood Search with 3-9 orders removed per iteration.

which were completed in approximately 0.8 seconds. The time is still too long, and the results seem too unstable for the current approach to be applicable as a subroutine.

An alternative approach would be to extend the neighbourhood structure of the existing implemented heuristics, by introducing the possibility of moving/swapping customers between routes, thus still solving the entire problem in one algorithm. The heuristics based on feasible operators (tabu search, simulated annealing and iterated local search) already depend on a combination of several operators, and this selection could thus be supplemented with an operator swapping visits between vehicles. A simple extension of the *complete-swap* operator would be sufficient to enable the algorithm to reach the entire solution space, but more sophisticated variations could be considered. An extension of the large neighbourhood search algorithm can be performed almost automatically. The removal operator must be updated to consider removal of orders from all vehicles, and the insertion operator similarly to consider insertion of orders into all vehicles.

2.5.1.2 Solutions

As described in the previous section, the large neighbourhood search code from Paper A can quite easily be modified to handle the multiple vehicle case. This has been done in order to provide some initial insights into the structure of the DVRPMS, and the results will be presented below.

In addition to the removal and insertion procedures, the savings algorithm used to construct initial solutions has also been updated. Since the vehicle tours to be constructed must all have the same length $m = R \times L$ (unlike a traditional solution for the VRP), the savings algorithm used for the TSP-version has been modified in the following way: Partial tours are merged, to gradually construct partial tours of increasing length. Whenever such a merge results in a tour that is too long, the new tour is split after the mth customer. This produces two tours, of which one has length m, and will thus not be considered again, and the other is shorter and will go on to be merged with other partial tours.

The 10 first problem instances from an extended data set[3] have been solved as DVRPMS instances using the modified LNS procedure, with 3 and 10 vehicles, and row capacities of 5 and 11. All instances are solved with a 3 row configuration.

The parameters of the LNS have been modified such that the number of orders to remove at each iteration is in the interval

$$\left[\max\left(2, \left\lfloor \frac{N}{9} \right\rfloor\right), \left\lceil \frac{N}{4.5} \right\rceil\right]$$

where N is the number of orders. The temperature interval for the acceptance criterion was set to

$$\left[\frac{z^{\mathrm{UB}}}{400}, \frac{z^{\mathrm{UB}}}{1600}\right]$$

where z^{UB} is the value of the initial solution supplied to the heuristic.

Due to the increased size of these problems, a further increase of the running time has been included for these tests, which have been conducted for running times 3 minutes, 10 minutes and 30 minutes. Table 2.6 reports the dimensions of the tested problems (vehicles $\times$ rows $\times$ row capacity), and the average solution value over 3 runs of each duration. The table also reports the overall best solution value found. For 4 of the 40 instances the best solution has been found in one of the short runs, and for another 5 the best solution has been found/matched for a 600 second run. This happens most frequently for the smaller instances ($3 \times 3 \times 5$, $N = 92$), but also for some medium sized instances ($10 \times 3 \times 5$, $N = 200$ and once for $3 \times 3 \times 11$, $N = 302$). This implies that a multi-start procedure might be beneficial, or that other measures should be taken

[3] `http://www.transport.dtu.dk/datasets/DTSPMS`

inst		n	z^{best}	180 s.	600 s.	1800 s.
R00	3×3× 5	92	1681	1.006	1.010	1.002
	10×3× 5	302	4207	1.081	1.027	1.028
	3×3×11	200	2533	1.062	1.053	1.034
	10×3×11	662	7263	1.098	1.036	1.015
R01	3×3× 5	92	1663	1.008	1.003	1.002
	10×3× 5	302	4342	1.057	1.035	1.006
	3×3×11	200	2663	1.032	1.025	1.008
	10×3×11	662	7286	1.102	1.052	1.016
R02	3×3× 5	92	1600	1.042	1.043	1.013
	10×3× 5	302	4264	1.061	1.033	1.016
	3×3×11	200	2553	1.070	1.038	1.028
	10×3×11	662	7343	1.107	1.040	1.011
R03	3×3× 5	92	1621	1.007	1.019	1.017
	10×3× 5	302	4410	1.038	1.015	1.012
	3×3×11	200	2651	1.043	1.024	1.020
	10×3×11	662	7387	1.072	1.040	1.018
R04	3×3× 5	92	1602	1.009	1.039	1.005
	10×3× 5	302	4320	1.063	1.043	1.008
	3×3×11	200	2617	1.062	1.034	1.011
	10×3×11	662	7444	1.110	1.080	1.020
R05	3×3× 5	92	1478	1.017	1.021	1.018
	10×3× 5	302	4312	1.055	1.045	1.010
	3×3×11	200	2557	1.037	1.032	1.014
	10×3×11	662	7318	1.103	1.043	1.028
R06	3×3× 5	92	1691	1.030	1.010	1.024
	10×3× 5	302	4332	1.061	1.036	1.017
	3×3×11	200	2623	1.042	1.027	1.016
	10×3×11	662	7120	1.110	1.085	1.015
R07	3×3× 5	92	1682	1.046	1.037	1.005
	10×3× 5	302	4361	1.034	1.021	1.011
	3×3×11	200	2648	1.086	1.042	1.028
	10×3×11	662	7376	1.094	1.042	1.013
R08	3×3× 5	92	1617	1.012	1.014	1.004
	10×3× 5	302	4239	1.049	1.029	1.015
	3×3×11	200	2559	1.043	1.017	1.027
	10×3×11	662	7421	1.107	1.049	1.011
R09	3×3× 5	92	1574	1.007	1.015	1.013
	10×3× 5	302	4267	1.091	1.040	1.007
	3×3×11	200	2591	1.053	1.033	1.022
	10×3×11	662	7204	1.102	1.067	1.008
avg.	3×3× 5	92	1647.7	1.018	1.021	1.010
	10×3× 5	302	4455.0	1.059	1.033	1.013
	3×3×11	200	2691.4	1.053	1.033	1.021
	10×3×11	662	7729.4	1.100	1.053	1.016

Table 2.6: Test results for the DVRPMS.

to avoid the algorithm getting trapped, such as increased neighbourhood size.

As can be seen from the table, instances with higher row capacity seem to be slightly more difficult than instances with more vehicles, even though they contain 50% more customers. This again supports the previous observations that the row capacity is really the one parameter that has the largest impact on the difficulty of solving a given instance. There does not seem to be a good explanation of why instances of $10 \times 3 \times 11$ seem easier to solve than $3 \times 3 \times 11$ – since the best solutions used for comparison are also determined by the runs reported in Table 2.6 it may be that there is simply less variation in the results for $10 \times 3 \times 11$, while these are in fact further away from their optimal solutions.

2.5.2 Further extensions

In this section some suggestions will be given for further extension possibilities for the DTSPMS. These will be limited to modifications that affect the handling of the special properties of the DTSPMS, such as modified loading patterns or alternative ways of handling the two graph setup. In addition, a range of more "standard" extensions could be performed, by including problem properties that are commonly applied to vehicle routing problems, such as time windows, dynamic order arrival, variable orders size, split deliveries, or load acceptance issues where not all orders must be served.

In practical applications, the balancing of load weight in the vehicle might also play a role, i.e. if a loading row on one side of the vehicle is almost full while one on the other side is still empty, this can affect the stability of the vehicle. Such considerations could lead to interesting additional constraints to be applied to the feasibility of a loading plan.

The descriptions given in the remainder of this section are rather brief, and focus on describing the modified problem that arises and its structural relation to the DTSPMS. Given the difficulty of solving real-life sized instances of the DTSPMS, and the fact that the DVRPMS may be more likely to be encountered in practice, it seems unlikely that any extension of the problem with further complications can be solved to optimality at current.

2.5.2.1 Alternative loadings

In real-life situations other loading patterns exist than the homogenous rows described so far. For example, depending on the shape, size, and properties of the transported items, a loading configuration has been encountered where one row is made up of items that are rotated 90° (in this case the items are on wheels, and can thus be handled from all sides). This results in a loading pattern with rows of varying length – in

this case 2 rows of length 9 and one row of length 21. Such a change can quite easily be incorporated in the existing heuristic and exact solution approaches, by adding a row index to the row capacity parameter.

Furthermore, some loading patterns in practical applications make use of tricks such as "turn the last two items of each row" in order to make better use of the total available loading. Such actions also require items that can be handled from the side, but occur in real-life. This modification can not immediately be adopted by the solution approaches presented here, but is mentioned as it presents an interesting twist to the loading problematic, and demonstrates a complication that may well be encountered in a real-life application of the DTSPMS.

2.5.2.2 Multiple transfer points

Instead of considering the start and end points (depots) of the routes as fixed points, these could instead be chosen among a number of points, reflecting that several terminals might be available for the transfer of the container(s) between the modes. This could for instance be relevant if the different terminals provided access to different modes for the long-haul transportation, and could then additionally lead to a connection between the choice of depot in the two graphs.

In this case the cost of the long-haul transportation would enter the objective function in some form, possibly leading to a multi-objective problem. This cost might express more than the simple travelling distance used in the DTSPMS itself, and could possibly include road tolls depending on route choice, or significant variations in travel time depending on mode choice. This would require the solution of the problem to take such multiple criteria into account.

Such a change could relatively easily be implemented in the heuristics already presented, by moderate modifications/extensions of the neighbourhood structures.

2.5.2.3 Pickup and delivery with LIFO and with multiple stacks – VRPPDLMS

The regular VRP with pickup and delivery under LIFO constraints (as described previously in Section 2.1.2) could be extended to have multiple stacks. The resulting problem could also be described as a DTSPMS with mixed pickups and deliveries.

This change would imply considerable modifications to the problem, both from the point of view of TSPPDL and DTSPMS, and probably no existing methods for either problem could easily be modified to accomodate this change.

2.5.2.4 Partial reloading

Another imaginable real-life setting of the DTSPMS could be obtained by allowing (partial) intermediate reloading of the container, albeit at a cost. This could happen if the transported items were fragile and required secure fastening, and their repositioning could thus be possible, but quite costly. This would pose the question of what price transportation companies would be willing to pay for the opportunity to move one item.

In this case there could be limits on the "reloading depth", such that only the k next items of each row would be candidates for rearrangement (since it might be too impractical to rearrange an entire row), or limited space could be available during reloading, imposing limitations on the total number of items to rearrange (allowing only a certain number of items to be placed outside of the vehicle at any time).

Allowing intermediate reloading would considerably change the structure of the problem, since the reloading decisions would need to be recorded. Such a change cannot be easily incorporated into the implemented heuristics, since these never considers the entire set of loaded items at a given point in time, but only a pair of items at their current loading position. At any time two items considered for swapping are selected such that the swap will be feasible, without making any considerations for the positions of the remaining items.

2.5.2.5 Hierarchical graphs

Finally, the DTSPMS or some of its concepts could be integrated in a larger intermodal framework, as a further extension to considering multiple transfer points. A problem could occur where a complete transportation chain is considered with several hiearchical levels, where the lowest level involves some loading constraints in line with those of the DTSPMS.

The transportation procedure could be as follows:

1. A number of small trucks pick up items from the customers.
2. At an intermediate pickup terminal all items are transferred from the smaller trucks to a container.
3. The container is transported to an intermediate delivery terminal, possibly using several modes to get there.
4. The items are transferred to a number of smaller vehicles.
5. Which perform the delivery of the items to the final destinations.

In this case the transfers between the levels might be performed with limited space available, meaning that the loading order would to some extent be reversed (items must be transferred directly between the smaller and larger trucks). This variant of the problem approaches the issues encountered during crossdocking operations.

Chapter 3

The Simultaneous Vehicle Scheduling and Passenger Service Problem

The second part of this Ph.D. has been focused on the problem that has been labelled the Simultaneous Vehicle Scheduling and Passenger Service Problem (SVSPSP), and is treated in the paper "The Simultaneous Vehicle Scheduling and Passenger Service Problem" (Appendix C). As the name indicates it is a variation of the vehicle scheduling problem (VSP), where passenger service is taken into consideration simultaneously with solving the traditional vehicle scheduling problem. Contrary to the classical VSP, this problem permits some moderate modifications of the timetable, which are intended to allow improvements of passenger service.

This chapter will first give a brief introduction to vehicle scheduling problems in general, and discuss relevant literature on problems which combine vehicle scheduling, (re-)timetabling, and passenger service considerations. It will then go on to introduce the SVSPSP and present some early results on solving the developed model with a standard solver, which were produced leading up to the work presented in Paper C. Finally, some supplementary comments on the paper will be given.

3.1 The Vehicle Scheduling Problem

The regular vehicle scheduling problem (VSP; see e.g. Desaulniers and Hickman [26] for a thorough introduction) is concerned with the assignment of (bus) trips/tasks to vehicles, in such a way that a given timetable is covered at the lowest possible cost. The timetable consists of a num-

ber of trips to be operated, and a set of vehicles are available to cover the trips. The purpose is to construct a set of vehicle schedules, such that each trip is operated exactly once, and all vehicle schedules are feasible (for any two trips a and b that are consecutive in a schedule, $starttime_a + duration_a + traveltime_{ab} \leq starttime_b$ must be respected, and possibly further restrictions may be imposed). The costs include deadhead costs incurred by empty travel between end points of consecutive trips, costs for transport to and from the depot (pull-in and pull-out), and possibly other elements. Travel times for the VSP are usually assumed to be fixed and deterministic. The number of available vehicles may be fixed, or its minimisation may be considered as part of the objective, by including a vehicle cost in the cost function. The problem may include vehicles available from several depots, leading to a multi-depot VSP (MDVSP), and in this case each vehicle must additionally start and end its schedule at the correct depot. Terminology used for the VSP is summarised in Table 3.1. Some slight modifications to this terminology will be introduced for the SVSPSP later in this chapter.

Trip	A *trip*, or *task*, has given start and end points, and a given departure time and duration.
Line	A *line* is a sequence of stops, such as the set of stops visited by bus 300. A line is often operated in both directions. For some lines there may be trips which only visit a subsequence of the stops, but still belong to that line (short-turning).
Trip incompatibility	Two trips that can not be operated consecutively by one vehicle are said to be *incompatible*.
Timetable	The timetable is the set of all trips that must be operated.
Schedule	A vehicle schedule describes the plan for one vehicle for one day, i.e. a set of trips that are non-overlapping (in time) and compatible.

Table 3.1: Terminology for the VSP.

The Single Depot VSP (SDVSP) is a VSP where all vehicles are stationed at the same depot. A number of trips are given, each with a fixed origin, destination, starting time and duration. Furthermore, it is assumed for now that a fixed number of vehicles are available to cover the trips, with all vehicles starting at the depot, at a known location. The task is now to construct a set of schedules, such that all trips are covered on time, at the lowest possible cost.

The SDVSP can be illustrated by a time-space network, as shown in Figure 3.1. In the figure, each trip is shown as an arc to illustrate the temporal aspect, but for modelling purposes each of these trip arcs can be collapsed to one node. Each arc going from a white node to a black node corresponds to a trip. Similarly the main depot node is only included in the figure for illustrative purposes, and can be left out of the model,

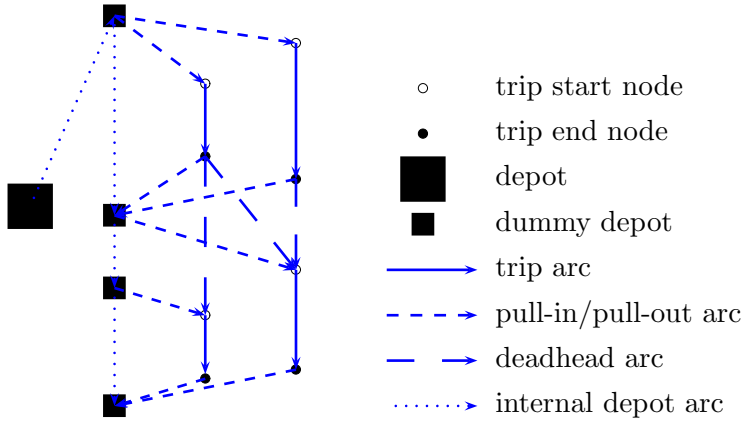


Figure 3.1: A time-space network illustrating the SDVSP.

along with its adjacent arc.

With the trip arcs collapsed, the problem can be expressed on a directed graph $G = (V, A)$ where the set of vertices consists of the trip nodes $i \in N$ and a set of dummy depot nodes $D = \{d_0, d_1, \ldots, d_{|D|-1}\}$, thus $V = N \cup D$. The set of arcs A in the graph is constructed such that all temporal constraints are satisfied; the existence of an arc (i, j) implies that $a_i + t_{ij} \leq a_j$, where a_i and a_j are the respective starting times of the two trips, and t_{ij} is the duration of trip i plus the deadhead time from the final destination of i to the origin of j.

The variable x_{ij} can then be introduced to express that the arc (i, j) is used, i.e. that trip j is performed immediately after trip i by the same vehicle (when $i, j \in N$, and with a suitable modification of the definition otherwise). The arc cost c_{ij} expresses the cost of using the arc (i, j), and may include any empty travel cost incurred when the destination of i is not identical to the origin of j. A possible daily cost of using a vehicle can be included in the costs $c_{d_0 \cdot}$, where d_0 is the first dummy depot node.

A mathematical model of the SDVSP can then be expressed as follows:

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{3.1}$$

s.t.

$$\sum_{i\in N} x_{ij} = 1 \qquad j \in N \tag{3.2}$$

$$\sum_{i\in V} x_{ij} = \sum_{i\in V} x_{ji} \qquad j \in V \tag{3.3}$$

$$\sum_{i\in N} x_{d_0 i} \leq v \tag{3.4}$$

$$x_{ij} \in \mathbb{B} \qquad (i, j) \in A. \tag{3.5}$$

where (3.1) is the objective function, containing the cost of all used

edges. Constraints (3.2) ensure that all trips are covered, and (3.3) are flow conservation constraints. In constraints (3.4), v is the number of available vehicles, to ensure that the maximum capacity is respected.

The SDVSP can be generalised to the multi-depot version (MDVSP), by introducing an extra set of dummy depot nodes for each additional depot, and duplicating each trip node for each depot (if all depots can not serve all trips some duplicates are omitted).

The SDVSP can be solved in polynomial time as a minimum cost flow problem, however the multi-depot version is NP-hard (Desaulniers and Hickman [26]). The MDVSP can be solved as a multi-commodity flow problem, and good exact procedures exist for solving large real-life instances. A mathematical formulation of the MDVSP will not be given here, but can be found in Desrosiers et al. [29], or in Paper C.

3.2 The Simultaneous Vehicle Scheduling and Passenger Service Problem

When the MDVSP is solved in practice it occurs as part of a planning process, where the timetable to be operated has been determined in the preceding phase, and the purpose of the MDVSP is then to operate the proposed timetable at the lowest possible cost. However, when the vehicle scheduling problem is solved for a given timetable, it is possible that a better solution could be obtained by allowing minor adjustments to the timetable, which would in comparison be insignificant. In the traditional planning approach such modifications are not permitted. In contrast, the SVSPSP regards the input timetable as a guideline, which can be modified during the vehicle scheduling solution process. This initial timetable expresses the required level of service with regard to frequencies and visited stops, and changes are only allowed such that these properties are maintained. The re-timetabling is in this case effectively performed by generating a set of shifted departure times for each trip of the original timetable, such that each trip may use any of these alternative departure times.

During the re-timetabling performed by the SVSPSP, the transfer opportunities provided by the solution are also considered. Transfer quality/duration is typically an objective when the timetable is produced, and by including this objective in the re-timetabling phase, it is ensured that the new solution still maintains these qualities.

The two objectives, operating cost and passenger convenience, are not necessarily contradictory – since the service levels have already been determined – but not concordant either. By the integration of re-timetabling elements into the vehicle scheduling problem, it becomes possible to consider these two objectives simultaneously when solving the SVSPSP. The purpose of the SVSPSP is thus to attempt to integrate two

traditionally separated problem solving phases, to include both operating cost, as expressed by the VSP, and passenger inconvenience (expressed as transfer time) when determining the final time table.

The combination of operating cost and passenger considerations naturally leads to a model with multiple candidate objectives; namely cost and passenger waiting time. In the case presented here, the choice has been to convert waiting time to a monetary value, using the published value-of-time for public transport, and add this value to the monetary objective expressing the operating costs.

For the presentation of the SVSPSP, a terminology will be used which ressembles that of the VSP, with some necessary modifications. In particular, not all trips of the original timetable must be covered in the SVSPSP: some of them may be replaced by alternative departures. To this end the concept *metatrip* is introduced to describe a set of trips, of which exactly one must be selected. Thus for each trip in the VSP, there is a metatrip in the SVSPSP, and each metatrip represents the possible occurences in time of a given (VSP-)trip. An overview of the updated terminology with the modifications for the SVSPSP is given in Table 3.2.

Trip	A *trip* or *task* is a departure on a given line at a given time. In contrast to the regular VSP, not all trips must be operated in the SVSPSP. Each trip belongs to exactly one metatrip.
Line	Same definition as for the VSP: a line describes a sequence of stops, which may be operated in either direction.
Metatrip	A metatrip is a set of trips, of which exactly one must be operated. Thus there is a metatrip for each trip in the original, corresponding VSP. The trips contained within each metatrip all share line, origin, and destination, and are close to each other in time.
Timetable	A timetable is part of a feasible solution to the problem, and contains all trips that are covered by the solution. In the VSP the timetable of the final solution is identical to input data.
Trip incompatibility	For the SVSPSP two trips are said to be incompatible if they are not allowed to be used in the same solution. This is typically used to avoid two consecutive trips departing at too long or short intervals.
Schedule	Same definition as for the VSP, an operating plan for one vehicle for one day.

Table 3.2: Terminology for the SVSPSP.

The SVSPSP can be illustrated by expanding Figure 3.1, duplicating each pair of trip nodes (white and black) for each depot (as for the MDVSP) and additionally for each possible departure time. Unfortunately

this expansion of the graph makes in rather unsuited for graphical representation, even for small examples, and thus an actual illustration has not been attempted here.

The VSP is traditionally solved from an operator point of view, where service requirements (timetables) are given by a contract, and the goal is to provide the required level of service at the lowest possible cost. Thus the issue of passenger inconvenience is not considered at this stage, and the two objectives operating cost and passenger inconvenience are not faced by the same entity. The SVSPSP attempts to consider the planning problem from a societal perspective, where passenger convenience and operating costs can indeed be considered simultaneously. This is justifiable, since ultimately the interest of the authorities is to obtain the best possible service at the lowest possible cost. The use of such a tool could improve the negotiating position of either the service provider or the operator, since each would benefit from having an improved perspective on the issues addressed by the other party.

The purpose of the introduction of the SVSPSP has been to study the possibilities of including the passenger perspective when solving the operator's problem, to demonstrate potential benefits of such an approach, and to examine whether passenger conditions can indeed be improved at a reasonable cost, by including these considerations in the process.

While the SVSPSP allows for time-shifting of trips, there are still restrictions imposed on such shifts. These restrictions apply for any two consecutive departures on the same line, and state that the distance in time between two such departures must be within given lower and upper bounds – typically expressed in terms of the temporal distance between the lines in the original timetable.

When allowing time-shifting in order to increase passenger comfort, there are two types of transfer times that can be considered, which, from a modelling perspective, are quite different. The first type is the intermodal transfers experienced by passengers transferring between two modes where one has a fixed timetable, external to the model, and the other has a variable timetable, which will be modified by the model. For such transfers the cost of a given trip can still be calculated independently of all other trips, and the objective function remains relatively straightforward. The second type of transfer is intramodal; passengers transferring between two links that both have variable timetables. When such transfers are allowed, the cost of trips is no longer independent, and the objective function typically becomes quadratic.

3.2.1 Literature on related problems

This section will discuss a selection of literature from areas related to the SVSPSP, combining vehicle scheduling problems and timetabling problems, with a variety of other aspects included that are also relevant to

the SVSPSP. Only very few papers have been found which treat problems that closely resemble the SVSPSP, so a range of papers covering similar problems will also be presented. The broader areas of VSP and timetabling in general will not be covered in detail, instead focus will be on the different complicating constraints arising in different situations. The purpose of this section is not to be exhaustive, but rather to give an impression of the multitude of problems from real-life that have been treated in the literature, and some of the different problem aspects that they consider.

The Vehicle Scheduling Problem, in single-depot and multi-depot versions, has been treated in the literature for years. An extensive survey up to 1995 is given by Desrosiers et al. [29], and a review of recent applications of operational research in public transportation, including vehicle scheduling problems, can be found in Desaulniers and Hickman [26]. This review covers the different planning phases, from strategical over tactical and operational to real-time disruptions, providing an overview over applied methods. A recent review focused around transit and transfer opportunities is found in Guihaire and Hao [54], which covers the network design, frequency setting and timetabling, and in particular points out work that integrates several of these phases.

Passenger aspects of timetabling problems can be obtained in two ways: by using so-called *timed transfers* where several buses arrive simultaneously at a transfer location, and have a layover time that is long enough to allow passengers to transfer, or by coordinating the departure and arrival times such that buses do not have to physically meet, but passengers can still transfer (this only allows for one-way transfer between two buses, since one bus will typically leave before the other arrives). In larger systems with many potential transfer locations, the use of timed transfers is often not a viable alternative, as it requires a considerable layover for each transfer, and also requires a certain layover slack at transfers to safeguard against delayed connections.

The majority of papers dealing with issues related to timetabling are based on real-life applications, and most of the presented results are based on real-life datasets. This shows that much of the research is spurred by realistic situations, but unfortunately has the side-effect that the results are difficult to compare. Furthermore the applications are often highly influenced by local conditions and traditions, both regarding the types of constraints that are applied, and the choice of objectives that are considered most important. Each model and solution approach is tailor-made to a specific situation, and there is not consensus in the literature about a standard problem definition, as is the case for example for the VRPTW or MDVSP. This in turn makes it difficult to compare solutions, and even to apply models developed by others.

Problems considering applications from the railway industry display some characteristics that differ from problems related to bus operation. Since trains often share the same tracks, collisions must be avoided. Further-

more, the use of periodic timetables seems to be more popular in the rail literature than in the bus literature – this is probably also caused by the track sharing constraints that are imposed on trains. However, e.g. underground systems or other local commuter train systems are sometimes operated without track sharing, thus avoiding the resulting safety issues. The literature discussed later in this section will be focused around problems concerned with bus scheduling, but will also include a few papers from the rail literature, in particular where some of the common, distinctive features are absent.

As mentioned, two different types of transfers are considered in planning problems: 1) external/intermodal transfers, between two modes where only one of them is controlled by the optimisation, while the other is external and fixed (typically transfer between a bus and a train, but transfers between an express bus and a city bus, or intercity and local trains may also occur), or 2) internal/intramodal transfers, between two trips that are both under the control of the optimisation procedure. The latter is typically the one considered in the literature where transfer opportunities are discussed, whereas the former naturally occurs more often in multimodal transportation, and has also received some attention in the literature, at times simultaneously with internal transfers. External transfer opportunities taken in isolation reduce to a matter of matching the timetable to the already given external lines, however, given that not every potential transfer can be covered, the choice of covered transfers remains.

There is a considerable amount of literature available on the topic of timetabling in general. The timetabling problems to be considered here will be a selection of recent problems that integrate timetabling with other planning phases, or exhibit other problem properties that are interesting from the point of view of the SVSPSP. Some of the included papers do not use solution methods from operational research, but have been included to demonstrate the variety of problems that exist. The considered timetabling problems generally optimise passenger service, with respect to some measure of transfer synchronisation, but there are a range of other properties which may or may not apply to each problem. These include:

- are resource considerations (the VSP aspect) included?
- is the service frequency predetermined or part of the optimisation?
- are intermodal or intramodal transfers considered? Many cover both, but some use only one or the other
- is stochasticity of travelling/arrival times considered?

The closest resemblance to the SVSPSP is probably that of the problem treated by Guihaire and Hao [53], which allows time shifting of a pre-determined timetable, and considers resource costs simultaneously with passenger inconvenience. Unlike the SVSPSP, the model uses timed transfers, and detects a transfer opportunity whenever two trips intersect within a certain allowed maximum time interval. Evenness of headway

is also considered, and a non-linear objective function is obtained as a weighted sum of these elements with vehicle cost. The problem is solved by means of an iterated local search approach, using an optimal SDVSP procedure to solve the vehicle assignment problem for the solutions that are constructed considering departure times of one or more trips. Experiments are carried out on a real-life dataset containing 25 lines with a total of 318 trips, and less than 30 train connections, using around 67 vehicles in the solutions found. The allowed running time was 10 minutes.

The case study by Liebchen and Möhring [72] also treats a problem similar to the SVSPSP, considering both internal and external connections in a hierarchical network involving two types of trains. The high-level trains have timetables that are set externally, and the task is to schedule the low-level trains to reduce waiting times at transfers both internally and to the higher level. The problem is modelled as a periodic event scheduling problem (PESP); an approach that seems particularly popular within train planning. The paper considers resource minimisation while determining a periodic schedule that provides the best possible transfer options, and also describes the process of adding additional constraints that are suggested by planners as the model is developed. Potential transfers between lines are assigned to a hierarchy, which is strongly, but not exclusively, based on the number of transferring passengers. The test results are viewed in this light, with focus on the most used transfers, and some tests completely disregarding 10–17% of transferring passengers, to examine the impact of such a reduction on the problem size. The problem is solved using a branch-and-bound algorithm with some heuristic elements, and tested on a data set which requires around 70 vehicles.

Another problem similar of the SVSPSP is described by Fleurent et al. [37]. The paper describes a software system which also incorporates resource usage and an adjustment of a pre-set timetable. Transfer quality is evaluated based on minimum, maximum and ideal waiting times, in order to calculate a synchronisation quality index. The paper does not give many details on solution method and results.

A larger group of papers exist which treat problems that do not contain the vehicle scheduling aspect, but cover problems that perform timetable modifications in order to improve transfers, possibly including a limit on the number of vehicles that are available. Chakroborty et al. [16] consider a timetabling problem for a bus transfer system, with the objective of minimising the total passenger waiting time (at the origins and at one transfer stop), under constraints of fleet size, stopping time, and headway. Thus the fleet size is considered as a constraint, rather than an objective. The problem is solved using a genetic approach, which is tested on a dataset with 3 routes and 30 vehicles, over a time horizon of 4 hours, with headway for routes with so-called “very high” demand being around 30 minutes. The network contains only one transfer stop,

however the authors state that their approach can rather easily be extended to handle multiple transfer points. A bus timetabling problem with intramodal transfers is also described by Cevallos and Zhao [15]. Their problem is based on shifting trips in an existing schedule in order to reduce transfer times. The authors consider stochastic bus arrival times, and use real-life data (40 bus lines, 255 transfer points, solution time up to 8 hours) to estimate the potential of the results.

Bookbinder and Désilets [8] consider the value of applying stochastic arrival times when considering timetable optimisation, and compare different measures of disutility, such as mean, mean squared, and variance. They present a range of interesting observations regarding the different measures, and also comment on the impact of using stochastic arrivals rather than deterministic.

In Ceder et al. [14] passenger service is measured by the number of simultaneous arrivals of trips, i.e. using timed transfers, rather than calculating the actual waiting times of transfers. The paper poses minimum and maximum restrictions on headways, but allows the individual departure times on each line to be set independently. The problem is solved using a heuristic procedure, tested on some small constructed examples, and one real-life example of moderate size.

Intermodal transfers are considered by Shrivastava and Dhingra [105], who consider a network of bus feeder routes each connecting to one of several railway stations. The paper considers both the bus frequencies and departure times. Transfers are considered feasible if the transfer time is larger than a value $T_{\min}$ (5 minutes) and any value larger than $T_{\max}$ (10 minutes) is penalised, as are overloaded vehicles. The number of buses required for each period of the day is also determined, and the problem is solved using a genetic algorithm. A similar system using feeder buses is considered by Chowdhury and Chien [17] where both bus and train timetables are determined, and both inter- and intramodal transfers are considered, but no transfers that are external to the considered problem. The objective is a combination of resource cost and user inconvenience (consisting of waiting time and in-vehicle time), and is obtained by adjusting slack time and headways. Vehicle arrival times are considered stochastic, and slack times are used to hedge against delays.

The so-called Timetable Synchronisation Problem (TTSP), which is a non-periodic, deterministic, railway planning problem, is solved by Wong et al. [113] by adjusting not only the headways, but also the drive times and dwell times of the trains. The system does not allow to insert or take out trains, so the available capacity is fixed within the planning horizon. A real-life data set from Hong Kong with 130–200 vehicles is considered, using a 1 hour planning horizon during rush hour, and a 2 hour planning horizon in non-rush hour.

Additionally, a variation of the MDVSP has been considered in the literature, which allows time shifting of the tasks in order to improve the

scheduling solutions. This variation has been referred to as the MDVSPTW, since it assigns a time window to each task, in which it must be performed. This problem does not include passenger considerations or transfers opportunities at all, and has been treated by Mingozzi et al. [79], Desaulniers et al. [27], and more recently Hadjar and Soumis [55].

Finally, van den Heuvel et al. [60] consider a problem that combines timetabling with vehicle scheduling for a heterogenous fleet, e.g. allowing for a demand of a given size to be covered by a combination of buses of different sizes, as long as the total capacity covers the total demand. The timetabling process does not consider transfer opportunities. A model is presented which allows for one trip to be covered by e.g. either one large vehicle, or two smaller vehicles. Several models are developed allowing various degrees of freedom when assigning buses to trips. The models are solved iteratively, by performing small changes to the timetable at each iteration, while solving the vehicle scheduling flow problem. The models are tested for weekdays, Saturdays, and Sundays, using real-life data with passenger counts for each trip.

To summarise, it can be concluded that a variety of papers have been published covering different aspects of the SVSPSP, but that no literature seems to exist, which covers the exact same flavour of the problem.

3.2.2 Mathematical formulation

A slightly simplified version of the SVSPSP can be formulated as a network flow model similarly to the MDVSP. Such a model will cover the vehicle scheduling aspect and the intermodal tranfers, but not the intramodal transfers. The cost of intermodal transfers only depends on the departure time of the trip itself, and all transfer times can be calculated a priori, based on the fixed timetables of the transfer mode. The costs of intramodal transfers however, depend on each of the two involved trips, potentially involving more than two metatrips, and are thus considerably more demanding to model mathematically.

For the SVSPSP, the simple, purely intermodal, model named SVSPSP^0 was first developed and implemented in CPLEX to be tested on some smaller instances, and secondly a large neighbourhood search heuristic was implemented for the complete problem, including the intramodal transfers. This two-stage approach was chosen for two reasons: 1) it was doubtful whether the exact model could be solved to optimality for real-life instances, and 2) the intramodal transfers are an important feature of the real-life problem, however adding this to the model would considerably increase the complexity of the model. The first, exact part of this process will be presented briefly in this section, while the heuristic solution has been described in detail in Paper C.

The initial, intermodal model SVSPSP^0 is presented in Paper C and stated below. It provides an important initial insight to the SVSPSP, and

a first approach to the declared purpose of simultaneously considering vehicle scheduling issues and passenger transfers.

The nodes of the graph represent either trips or dummy depots, and the variable x_{ij}^d is used to express that node j is covered immediately after node i by a vehicle belonging to depot d. The cost coefficient c_{ij}^d expresses the cost of doing so. The set of metatrips is Ω, and Φ is the set of sets of *incompatible* trips, which for the SVSPSP means trips that are not allowed to be in a solution together. Each set in Φ is a set of trips that are mutually incompatible. Incompatible trips typically arise from headway restrictions, and help ensure a certain level of passenger service. K is the set of depots.

$$\min \sum_{k \in K} \sum_{(i,j) \in A^k} c_{ij}^k x_{ij}^k \tag{3.6}$$

$$\sum_{i \in M} \sum_{k \in K} \sum_{j \in V^k} x_{ij}^k = 1 \qquad M \in \Omega \tag{3.7}$$

$$\sum_{i \in \phi} \sum_{k \in K} \sum_{j \in V^k} x_{ij}^k \leq 1 \qquad \phi \in \Phi \tag{3.8}$$

$$\sum_{j \in N} x_{n+k,j}^k \leq v^k \qquad k \in K \tag{3.9}$$

$$\sum_{i \in V^k} x_{ij}^k - \sum_{i \in V^k} x_{ji}^k = 0 \qquad k \in K, j \in V^k \tag{3.10}$$

$$x_{ij}^k \in \mathbb{B} \qquad k \in K, i, j \in N \tag{3.11}$$

$$x_{ij}^k \in \mathbb{N} \qquad k \in K, (i,j) \in A^k \backslash (N \times N). \tag{3.12}$$

The objective function (3.6) expresses the cost of the solution. This includes both the operating costs, as known from the VSP, and the intermodal transfer costs which are contained in the arc costs. Constraints (3.7) ensure that exactly one trip from each metatrip is covered, and (3.8) ensure that all incompatibilities are respected. (3.9) limits the number of vehicles available at each depot, and finally (3.10) are flow conservation constraints.

As stated previously this model does not cater for intramodal transfers between buses, since the costs as expressed here can only cover passengers transferring to or from a connection with a fixed timetable.

3.2.3 CPLEX Solution Results

The model (3.6)–(3.12) has been implemented in Java using ILOG Concert Technology with CPLEX, and run on an Intel Pentium 4, 2.8 GHz, with 2GB RAM, running Windows XP. Due to the size of the real-life instances, the model was only tested for some smaller, partial instances.

The size of the instances that have been tested, have been controlled by several parameters:

- number of lines
- number of depots
- number of metatrips considered per trip
- length of the day

Unfortunately, modifying the number of lines or length of the day does not only affect the size of the problem, but also some of its characteristics. The choice of lines to include in each instance affects the number of possible transfers, as well as the number of trips operated per hour. The tests of the initial model have been focused on the most central lines, which means that there is a quite large amount of traffic, and a high number of possible transfers, particularly intermodal transfers. All calculated days have been started early in the morning, but for some runs the length of the day has been reduced, e.g. by excluding all trips that depart later than 18.00, rather than going on to the regular end of schedule around midnight (there is not round-the-clock service on any of the lines considered for this data set). Thus the most busy period (morning rush hour) has been included for all tests, and only the less busy evening period has been left out. This still results in a reduction of the model size, however the number of vehicles used in total can be expected to be unaffected when the time horizon is adjusted this way.

l	d	s	h	Time/s	Root/s	Rows	Columns	Non-zeroes	Obj.
3	2	5	24	395	352	$8.9{\cdot}10^3$	$773{\cdot}10^3$	$3.40{\cdot}10^6$	$798{\cdot}10^3$
3	3	3	24	409	288	$8.0{\cdot}10^3$	$426{\cdot}10^3$	$1.68{\cdot}10^6$	$819{\cdot}10^3$
3	3	3	18	223	200	$6.6{\cdot}10^3$	$377{\cdot}10^3$	$1.49{\cdot}10^6$	$796{\cdot}10^3$
3	3	5	18	970	903	$10.1{\cdot}10^3$	$1.03{\cdot}10^6$	$4.52{\cdot}10^6$	$776{\cdot}10^3$
3	3	5	24	1546	1354	$12.3{\cdot}10^3$	$1.16{\cdot}10^6$	$5.10{\cdot}10^6$	$798{\cdot}10^3$

Table 3.3: Initial results of CPLEX for SVSPSP^0.

Table 3.3 shows the dimensions of the tested problem, with l lines, d depots, s trips per metatrip and h being the latest departure considered (h is not directly the length of the planning period, since operation starts around 05.00 and not at midnight). For each tested instance the table reports the solution time in seconds, the solution time for the LP-relaxation of the root node, and the size of the reduced MIP as reported by CPLEX. All 5 instances are solved in the root node, with an objective value equalling the value of the LP-relaxation.

As it turned out the biggest hindrance for solving the model was memory, and the test computer with 2GB RAM failed for larger instances. Some CPLEX parameter settings to possibly alleviate this problem, such at depth-first search, were tested, but the model could still not be solved using these settings either.

In particular the results of the test runs indicated that a potential reduction in waiting time could indeed be obtained by following the integrated

solution path suggested with the formulation of the SVSPSP.

Since the SVSPSP^0 model was unable to include the intramodal transfers, it was decided not to pursue this solution path any further.

3.3 Paper: Heuristic Solution Approaches for the SVSPSP

Paper C presents a heuristic solution approach to the SVSPSP which shows promising results on the real-life sized test instances, and this section will provide some additional comments on issues that are relevant to the paper. It is recommended that the reader read the paper before reading the remainder of this section.

Some simplifying assumptions are made in the paper, and some choices are made on how to regard passenger behaviour. In particular, it has been assumed that all passengers read the published timetable, and that half of the passengers make their journey based on a desire to be at a given destination at a given time, and select their departure time such that this target is obtained. The other half will wish to leave their origin at a certain time, and have no preference regarding arrival time. This imposes some complications on the model, by requiring the use of both y and z variables in the formulation (C.13)–(C.21), and also complicates the implementation of the heuristic solution approach. However, it has seemed potentially interesting to also allow some transfers to be defined by the embarking connection, given that there are certainly some passengers who plan their journey this way. This is opposed to the more traditional approach of only considering transfers based on the disembarking connection (passengers arrive at a terminal, and depart by the first valid departure). It could be considered whether the distribution between these two types of planning should indeed be equal, or if some smaller percentage should be used for embarking passengers.

A simplification of reality can be found in the assumption that passengers' route choice is unaffected by the operated timetables. This is the traditional approach within operational research, but traffic researchers from other areas will disagree with this simplification. Solving an integrated problem, which optimises under consideration of route choice models would definitely present an interesting direction for future work. Furthermore, e.g. the number of passengers on a vehicle will affect its travel time, due to increased (dis)embarking times, and stochastic travel times in general is also a topic that could be included in the considerations. Finally, the value-of-time that has been used to calculate the value of saved time for the passengers, expresses the value as perceived by the passengers. It is not obvious that this value corresponds to the value of passengers' time as experienced by the service provider. However, as the presentation of the SVSPSP at this stage has been focused on the development of the model and proof-of-concept of the method, these values

have been adopted directly.

The issue of obtaining travelling data for passengers is another interesting topic. In the paper, travelling patterns have been expressed by a estimated number of passengers embarking/disembarking at every transfer point, and an estimated distribution of these passengers among the available connections. This approach has made the transfers easier to handle, since it did not require knowledge of origin-destination pairs, and did not check for consistency of data in the form of "x passengers disembark A to embark B" and "y passengers embark B coming from A", where $x = y$ would be a natural assumption, but is neither assumed nor verified in this form of the model. At present it has been estimated that the uncertainty on the travel data is significantly larger than this inconsistency.

With the advent of electronic ticketing systems, it must be expected that much better data will become available for planning problems like the SVSPSP. At least with respect to the actual travels and transfers on a given day, it should become possible to obtain accurate, observed data, which would be a significant improvement over the current situation, where data are often based on estimates and qualified guesswork. However, for electronic ticketing to become a reliable source of data, it should be used consistently by all travellers, and this situation still lies several years into the future in Denmark. Until then, the results of the paper show that there is a potential gain from simultaneous optimisation, but in order to realise this effect in practice, an improvement of the data quality is needed.

Finally the effects of having a "memorable" timetable have been briefly touched upon in the paper, and is an issue that deserves further attention before any solution can possibly be implemented in real-life. Many timetables today consist of several periods of periodic departures (e.g. "from 15.02 a bus departs every 10 minutes until 18.32"), which makes the timetables (or parts thereof) easier to memorise for passengers, and is a feature of user-friendliness that is quite independent of the cost of operation of the required vehicles. This memorability of solutions has not been included as a feature of the solution procedures tested so far, and the results from the paper show that this causes a drop in the memorability of the produced solutions. Two obvious ways to prevent this would be either by adding a component to the objective function which somehow penalises departures at irregular intervals, or by only allowing timeshifting to be performed on pre-defined blocks of metatrips, such that the headways within the blocks are unchanged.

Chapter 4

Conclusion

This thesis has covered two different problems, which both occur in an intermodal setting and arise in situations where multiple objectives may need to be considered. One problem comes from the world of freight transportation, while the other comes from public transport. To the best of the author's knowledge, neither of the problems has been described previously in the literature, similar to the versions presented in this thesis.

The Double Travelling Salesman Problem with Multiple Stacks (DTSPMS) is an interesting problem, which combines routing and loading aspects. It is concerned with determining the shortest possible routes for performing a set of pickup and delivery operations, under a set of loading constraints. These loading constraints pertain to the vehicle which is used for the transport of the goods, which does not allow random access to the loaded items, but only allows access in a LIFO stack fashion. Furthermore the items that are transported are of such type that the loading space of the vehicle can be divided into several independent LIFO stacks. Pickup and delivery problems with LIFO constraints have already been treated in the literature, but the introduction of several independent stacks is new. The idea to use multiple loading stacks has been inspired by a real-life case from the transport industry, and the introduction of this new problem has already inspired several researchers to either work on either the DTSPMS, or to integrate the usage of multiple loading stacks into other problems.

An interesting property of the problem that emerged is that the total number of orders to be handled is not as important, in terms of solution difficulty, as the capacities of the LIFO stacks, particularly regarding the difficulty of obtaining an optimal solution to the problem. The number of stacks that are available has a much smaller impact on this difficulty.

For the solution of the DTSPMS, a selection of well-known metaheuristics have been implemented, and in particular the large neighbourhood search approach has demonstrated good results on instances of real-life size. Results presented here, and by others, have indicated that the complexity of the solution space is such that heuristics applying only small modifications to the solution in each iteration may find it difficult to obtain good results. So far the best results have been obtained by heuristics that use more extensive modifications, such as large neighbourhood search, or that allow the use of intermediate infeasible solutions.

Some exact solution approaches for the DTSPMS have also been developed and implemented, all based on a branch-and-cut approach using CPLEX. Different modelling approaches have been considered and tested, and particularly good results were obtained using a decomposition approach, which divides the problem into a routing and a loading part, with the routing problem being solved as the master problem, and the loading feasibility problem solved as a subproblem, to identify loading infeasible subpaths, and exclude these from the solution. A long way still remains before real-life-sized instances can be solved to optimality, but the size of the problems that can be solved has been improved noticeably, in comparison to the initial formulation.

The Double Travelling Salesman Problem with Multiple Stacks is, as the name suggests, a single vehicle problem, however it can quite easily be extended to the multiple vehicle case: The Double Vehicle Routing Problem with Multiple Stacks. The best of the implemented metaheuristics for the DTSPMS has been extended to the DVRPMS case, and some results are presented, along with other suggestions and ideas for further development and extension of the problem.

The other problem which has been introduced with this thesis is the Simultaneous Vehicle Scheduling and Passenger Service Problem (SVSPSP). This is a problem from public bus transport, which combines aspects of timetabling and vehicle scheduling. Usually these two problems are handled separately, but the approach suggested here integrates (re-) timetabling into the vehicle scheduling problem, potentially allowing for better vehicle schedules for the operator, and improved transfer opportunities for the passengers. The results are promising, and indicate that passenger inconvenience can be decreased at the cost of a slight increase in empty vehicle mileage.

The Simultaneous Vehicle Scheduling and Passenger Service Problem includes several different aspects. Apart from the usage cost and empty mileage incurred by the vehicles, it considers passenger transfer times, both between the buses whose service are controlled by the algorithm, and in relation to external timetables, which are in this case trains.

Initially, a mathematical model of the problem was implemented using CPLEX. This model was based on a flow formulation of the vehicle scheduling problem, which was modified to handle the temporal shifts

and the transfer to external modes. This model ignored the internal transfers between buses. The results obtained from testing this model were not discouraging, but the ability to handle internal transfers would require considerable modifications, and were thus decided against.

In order to solve the problem for the complete dataset that was intended from the outset, namely the express bus network of Copenhagen, a heuristic solution approach to the SVSPSP was chosen, handling external and internal transfers, as well as timeshifts and vehicle costs. A large neighbourhood search algorithm was implemented, and the results were promising, both with regard to the ability of the algorithm to solve the problem, and with regard to achievable savings in passenger waiting time.

Overall, the problems presented in this thesis have provided valuable input to the research society. The presented solution approaches have all shown promising results, and can hopefully provide a platform for future work by myself and others.

Appendix A

The Double Travelling Salesman Problem with Multiple Stacks – Formulation and Heuristic Solution Approaches

Authors:
Hanne L. Petersen, Oli B.G. Madsen.

Abstract:
This paper introduces the Double Travelling Salesman Problem with Multiple Stacks and presents four different metaheuristic approaches to its solution. The Double TSP with Multiple Stacks is concerned with determining the shortest route performing pickups and deliveries in two separated networks (one for pickups and one for deliveries) using only one container. Repacking is not allowed, instead each item can be positioned in one of several *rows* in the container, such that each row can be considered a LIFO (last in, first out) stack, but no mutual constraints exist between the rows. Two different neighbourhood structures are developed for the problem and used with each of three local search metaheuristics. Additionally some simpler removal and reinsertion operators are used in a Large Neighbourhood Search framework. Finally some computational results are given along with lower bounds on the objective value.

Errata A few minor corrections to the published paper have been made in the version included here:

- the index τ just before the section on *Route-Swap* is now G.
- the text at the end of sec 4.4 now reads "increased values of L" and "$R = n$", i.e. "will deteriorate with increased values of L, since the nS bound is exactly the optimal solution for $R = n$".
- some slight inaccuracies were discovered in the top half of column SS of Table A.1, and have been corrected. This does not affect any results or conclusions of the paper.

In addition, some minor typographic and aesthetic changes have been made.

A.1 Introduction

As congestion is an ever-growing problem on the roads all over the world, intermodality is playing an increasingly important role in the transportation of goods. Furthermore, the complexity of the resulting planning problems presents additional requirements to the tools available to planners.

The project that forms the basis of this paper was initiated in cooperation with a company producing computer software systems for operation and fleet management in small and medium-sized transportation companies. The software company encountered this problem at one of its prospective customers, and the problem is intriguing in that it does not seem to have been treated previously in the literature, at the same time as it is conceptually simple.

The Double Travelling Salesman Problem with Multiple Stacks (DTSPMS) is concerned with finding the shortest routes performing pickups and deliveries in two separated networks/regions. The problem permits neither repacking nor vertical stacking, instead the items can be packed in several *rows* (horizontal stacks) in the container, such that each row must obey the LIFO (Last-In-First-Out) principle, while there are no mutual constraints between the rows.

In the DTSPMS a set of orders is given, each one requiring transportation of one item from a customer in the pickup region to a customer in the delivery region, i.e. each order contains a pickup customer and a delivery customer for one item. The items are required to be boxes/pallets of identical dimensions and each region has a depot. The two regions are far apart, and thus some long-haul transportation is required between the depots. This long-haul transportation is not part of the problem considered here. All pickups and deliveries must be carried out using the same container, which cannot be repacked along the way, and items in the container can only be accessed from the opening in one end of the container. Hence the problem to be solved consists of determining the

shortest Hamiltonian tour through each of the networks, in such a way that a feasible loading plan exists. No time windows are considered in this problem.

In practice this situation can occur when the container is loaded onto a truck to perform the pickup operations, then returned by that truck to a local depot/terminal where it is transferred onto a train, ship or another truck, which then performs the long-haul transportation. Upon arrival at the depot/terminal in the delivery region, the container is again transferred to a truck, which carries out the actual deliveries. The terminals only have facilities to perform container movements, and do not offer any opportunities for opening or repacking the container.

It is assumed that each order consists of exactly one item, thus if one pickup or delivery location is shared by several orders the corresponding node in the graph will be duplicated.

To state the problem more formally two weighted complete graphs (V^G, E^G), $G \in \{\mathrm{P}, \mathrm{D}\}$ are given for pickup (P) and delivery (D) respectively, and the purpose is to find a Hamiltonian tour through each graph, such that the sum of the weights of the edges used is minimised. Each graph G has a depot node v_0^G, $G \in \{\mathrm{P}, \mathrm{D}\}$, customer nodes $v_1^G, \ldots, v_n^G$, and symmetric edge costs c_{ij}^G. A set of n orders $\{1, \ldots, n\}$ is given, where order i must be picked up at $v_i^{\mathrm{P}} \in V^{\mathrm{P}}$ and delivered at $v_i^{\mathrm{D}} \in V^{\mathrm{D}}$. Finally, $V_C^G = V^G \backslash \{v_0^G\}, G \in \{\mathrm{P}, \mathrm{D}\}$ denotes the set of customer nodes in graph G. Whenever superscript G is used in the following it will indicate a distinction between pickup and delivery, $G \in \{\mathrm{P}, \mathrm{D}\}$.

The edge costs are assumed to be symmetric throughout this paper, but the presented algorithms do not rely on this property, and can also handle problems with asymmetric edge costs.

Throughout the paper it will be assumed that the number of orders n equals the number of loading positions. The loading container will have R rows, each of length L, and thus $n = R \cdot L$. The total number of nodes in the problem is $2n + 2$.

The connection between the two tours to be found is given by the loading of the container. Since no repacking is allowed, the only items that can be delivered "next" at any time during delivery are the ones that can be accessed from the opening of the container. This implies that the loading is subject to LIFO constraints.

However, in the DTSPMS there is no LIFO ordering for the container as a whole. Rather, it contains several loading rows, each of which can be considered a LIFO stack, but all rows are independently accessible.

In real life the items to be transported would typically be standardised Euro Pallets, which fit 3 by 11 on the floor area of a 40-foot pallet container, providing three independent loading rows.

A solution to a given problem consists of a *pickup route*, a *delivery route*,

and a *row assignment*, which for each item indicates which loading row it must be placed in. A row assignment only gives the row that each item should be placed in, and does not indicate which position the item will occupy in that row. Given a route (pickup or delivery) and a row assignment, one can construct the *loading plan*, which gives the exact position of each item inside the loaded container.

The problem may at first glance seem purely theoretical, since the extra mileage incurred by not being able to repack may seem prohibitive. However the problem has been encountered in real-life applications, where this extra mileage is justified by the wages stemming from handling and requirements to comply with union restrictions (the driver is not allowed to handle the goods).

Special cases of the DTSPMS occur when the number of loading rows is equal to 1 or to the number of orders n. In both cases the problem of finding a row assignment for the solution becomes irrelevant.

In the single row case the pickup route will strictly dictate the delivery route (or vice versa), and the two routes will be exact opposites. In this case the problem can be solved by adding the transposed distance matrix of the delivery graph to the distance matrix of the pickup graph, and solving a regular TSP for the resulting distance matrix.

Conversely, when the number of loading rows equals the number of orders n, the two routes do not impose any restrictions on each other and the optimal solution to the problem consists of the optimal solutions to the two independent TSPs.

The DTSPMS as described here is a combination of the travelling salesman problem (TSP) and pickup and delivery problems (PDPs) and does not seem to have been treated previously in the literature, however early presentations of the work presented in this paper have inspired additional work presented in Felipe et al. [34], which uses several new operators, along with the ones presented here, in a Variable Neighbourhood Search (VNS) framework.

Although being concerned with pickups and deliveries, the DTSPMS differs significantly from the "regular" PDP as described in e.g. Cordeau et al. [20] and Desaulniers et al. [28], and earlier in Kalantari et al. [64] and Savelsbergh and Sol [100]. A number of variations of the PDP have been described in the more recent survey Parragh et al. [85, 86].

The main additional complication is the availability of multiple LIFO loading rows and thus the need to present a loading plan as part of the solution. The regular PDP with LIFO ordering (one stack only) has been treated using both heuristics Carrabs et al. [12] and exact methods Cordeau et al. [23] and Carrabs et al. [11].

In the regular PDP it is necessary to make sure that each pickup is performed before the corresponding delivery. This is automatically ensured in the DTSPMS, since all pickups are performed before all deliveries.

Furthermore when capacity constraints are present in a PDP these must be checked for every node that is visited. In the DTSPMS all items will need to be kept in the container at the same time, and therefore it would suffice to check the capacity of the full vehicle if any kind of capacity constraints were present (which would not happen in the plain DTSPMS, but could occur with an extension to multiple vehicles, or if not all orders need to be served).

Apart from the regular PDP, another class of problems that show similarities with the DTSPMS is the TSP with Backhauls (TSPB) (cf. e.g. Toth and Vigo [110]). Here the property that "all pickups lie before all deliveries" is preserved, however there is no longer any constraints tying a pickup to its corresponding delivery.

The DTSPMS is a special case of the generalised Pickup and Delivery Problem with Loading Constraints in 2 dimensions. Routing problems with more general loading constraint are described in Iori et al. [61], Gendreau et al. [45] and Gendreau et al. [44].

The Multi-Pile VRP (MPVRP), is a problem somewhat similar to the DTSPMS, combining routing and loading, using several available piles/stacks. Doerner et al. [30] solve the MPVRP using tabu search and ant colony optimisation. The MPVRP is a generalisation of the DTSPMS, with varying dimensions of the transported items (leading to overlap between the stacks/piles).

The paper is organised as follows: First a mathematical formulation of the problem is presented in Section A.2 and some comments are made on its implementation in GAMS/CPLEX. Next, four different heuristic solution approaches are presented in Section A.3, with emphasis on the developed neighbourhood structure that is based on the structure of the problem, and is used for the first three approaches. Finally, Section A.4 describes the implementations and gives some computational results, and Section A.5 concludes on the described heuristic solution approaches and gives some suggestions for future work on the DTSPMS.

A.2 Mathematical Formulation

The DTSPMS can be modelled as a binary integer programming problem with variables

$$x_{ij}^G = \begin{cases} 1 & \text{if edge } (i,j) \text{ is used in graph } G, \\ 0 & \text{otherwise,} \end{cases} \qquad i,j \in V^G,$$

$$y_{ij}^G = \begin{cases} 1 & \text{if } v_i^G \text{ is visited before } v_j^G, \\ 0 & \text{otherwise,} \end{cases} \qquad i,j \in V_C^G,$$

$$z_{ir} = \begin{cases} 1 & \text{if item } i \text{ is placed in row } r, \\ 0 & \text{otherwise,} \end{cases} \qquad \begin{array}{l} i \in V_C^G, \\ r = 1, \ldots, R. \end{array}$$

Again $G \in \{\mathrm{P}, \mathrm{D}\}$.

The objective function can then be expressed as:

$$\min \sum_{\substack{i,j \in V^G \\ G \in \{\mathrm{P},\mathrm{D}\}}} c_{ij}^G x_{ij}^G \tag{A.1}$$

The constraints can be stated as follows:

$$\sum_i x_{ij}^G = 1 \qquad \forall j \in V^G \tag{A.2}$$

$$\sum_j x_{ij}^G = 1 \qquad \forall i \in V^G \tag{A.3}$$

$$y_{ij}^G + y_{ji}^G = 1 \qquad \forall i, j, G, i \neq j \tag{A.4}$$

$$y_{ik}^G + y_{kj}^G \leq y_{ij}^G + 1 \qquad \forall i, j, k, G \tag{A.5}$$

$$x_{ij}^G \leq y_{ij}^G \qquad \forall i, j, G \tag{A.6}$$

$$y_{ij}^{\mathrm{P}} + z_{ir} + z_{jr} \leq 3 - y_{ij}^{\mathrm{D}} \qquad \forall i, j, r = 1, \ldots, R \tag{A.7}$$

$$\sum_r z_{ir} = 1 \qquad \forall i \tag{A.8}$$

$$\sum_i z_{ir} = L \qquad \forall r = 1, \ldots, R \tag{A.9}$$

$$x, y, z \in \mathbb{B}. \tag{A.10}$$

where i, j and k are in V_C^G unless otherwise stated, and G is always in $\{\mathrm{P}, \mathrm{D}\}$.

Constraints (A.2) and (A.3) are flow conservation constraints, stating that one unit of flow must enter and exit each node.

Constraints (A.4) ensure that for each pair of nodes (i, j) a precedence variable must be set, i.e. either i is visited before j or j before i. (A.5) express that if i is before k and k is before j, then i must necessarily be visited before j and constraints (A.6) ensure that if the edge (i, j) is used, then the according precedence variable is set (i is visited before j).

Constraints (A.7) express the LIFO constraints that are only relevant when two items are in the same row, i.e. if i and j are placed in the same row, and i is picked up before j, then i must be delivered after j (i may not be delivered before j).

Finally, (A.8) ensure that all items must be assigned to exactly one row, and (A.9) enforce the row capacity/length L.

The model has been implemented in GAMS 21.5 with CPLEX 9.1, on a UNIX system with 16 GB RAM/1200 MHz, which was able to solve problems for container sizes up to 2 by 5 or 3 by 4 within an hour of running time. Since the typical real-life instance is of size 3 by 11 and the time available for solving was limited, it was therefore decided to attempt to solve the problem heuristically.

A.3 Heuristic Solution Approaches

Since the mathematical model is unsolvable for problems of realistic size using a standard solver, a number of metaheuristic solution approaches have been considered for this problem. A survey of previous use of metaheuristics in vehicle routing problems can be found in [43].

Tabu Search (TS) has previously presented good solutions to vehicle routing problems, which are similar in nature to the current problem. Many variations of tabu search exist, however this paper will only consider the simple version, which always uses the best neighbouring solution, and has constant tabu tenure.

Simulated Annealing (SA) is another method that is well-known to provide good results to many different problems. Its advantage over TS is that it can move faster through a number of neighbourhoods, since it immediately chooses one neighbour at each iteration, rather than comparing several neighbours, and can thus complete a higher number of iterations in a given time. This comes at a cost of more randomised behaviour.

To conclude the traditional neighbourhood-based metaheuristics, a simple *Steepest Descent* approach has been implemented to determine how easy it is to locate good local optima from random starting points in the solution space. This has then been used in a simple Iterated Local Search (ILS) with random restarts and steepest descent used as the local search strategy.

Finally a *Large Neighbourhood Search* (LNS) algorithm has been implemented for the problem. LNS has in recent years showed good performance for VRP-like problems and seems promising when dealing with highly constrained problems, such as the one treated here.

The first three solution approaches all solve the problem by local search in a rather limited neighbourhood and are all based on an initial solution and some neighbourhood structure. Thus once one or more neighbourhoods have been developed, they can be reused for several approaches. Two different neighbourhoods have been developed here, both of which preserve feasibility of the solution, and, as it will be explained later, in combination the two can cover the entire feasible solution space. These two neighbourhoods have been implemented for use with each of the first three above-mentioned heuristic approaches.

The LNS approach is based on a combination of simpler operators, performing either deletion or insertion of orders. Since these operators are less problem-specific, this approach allows for the operators to be inspired by heuristic solutions to more general problems, such as the TSP or VRP.

Since the company that introduced the problem were interested in the running times that would be experienced by the customers, wall clock

times was chosen as the stopping criterion. This additionally ensured that the results from the different algorithms would be directly comparable. A 10 second interval was chosen to resemble online computations, while 3 minutes was expected to produce considerably better solutions within a duration that users would still be willing to wait for.

A.3.1 Initial Solution

A feasible solution to the problem can be found by solving the single-stack version of the problem. In this case the pickup and delivery routes must be exact opposites, and the solution can be obtained by adding the two distance matrices and solving a regular TSP on the resulting distance matrix. This problem has been solved using a savings algorithm as introduced by Clarke and Wright in [18] to solve the TSP, and this initial solution has been used for the TS, SA and LNS implementations, since these only need one initial solution to the problem.

For the ILS, a number of initial solutions were constructed by randomly generating an ordering of the items to use for the pickup route, and reversing this for the delivery route, thus all of the generated solutions are still based on feasible solutions to the single-stack problem. Loading rows were then assigned randomly by partitioning all orders evenly among the available rows.

A.3.2 Feasible Neighbourhood Structures

In this section the two operators that form the basis of the three initial solution approaches are introduced.

Route-swap

The first operator only performs changes to the routing of the two tours, and leaves the row assignment untouched. The neighbourhood consists of all possible exchanges of two items A and B in a route where they immediately follow each other. To ensure feasibility of the resulting solution it is necessary during this operation to consider whether the two items are placed in the same loading row. These two cases can be seen in Figure A.1 (top resp. bottom). From left to right the figure shows pickup route, loading plan and delivery route, before and after performing a *route-swap*.

If A and B are placed in the same row (top of Figure A.1), then they will be neighbours in this row and their positions in the loading plan of the container will be swapped by swapping their pickup order. Consequently their positions must also be swapped in the delivery route (even when A and B are not consecutive in this route).

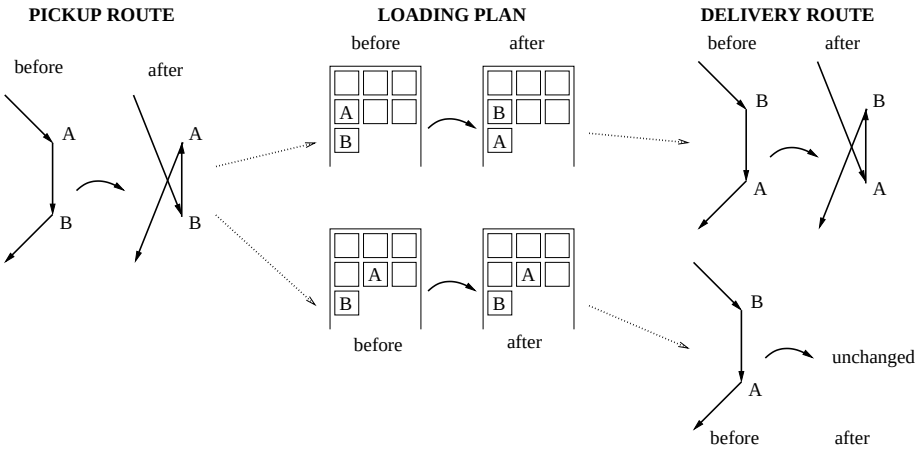


Figure A.1: An illustration of *route-swap*.

If A and B are placed in different loading rows (bottom part of Figure A.1), then, since they are consecutive in the pickup route, this means that when $v_{\mathrm{A}}^{\mathrm{P}}$ is visited, $v_{\mathrm{B}}^{\mathrm{P}}$ will be next in line and since A and B are in separate rows, then the loading position of B will also be accessible (and be the last accessible position in its row), and thus the positions of A and B in the final loading plan can remain unchanged when swapping the pickup order, and so can the delivery route.

Both of these arguments can be reversed when the operator is used on the delivery route.

The entire neighbourhood can be traversed by considering all values of $i = 1, \ldots, n-1$ for both routes G, thus the size of this neighbourhood is $2n - 2$, i.e. $O(n)$, where n is the number of orders.

Complete-swap

The operator *complete-swap* is focusing on the row assignment, while only updating the routes to maintain feasibility. It considers any pair of items that are currently assigned to different rows (regardlessly of routing), and swaps their positions in the loading plan and in each of the routes. This is illustrated in Figure A.2. The top half shows (part of) the two graphs and loading plan *before* performing the *complete-swap*, while the bottom half shows them after. A and B are the two orders that are swapped, while C is some order that is visited between the two, and would be blocking if the routes were not updated as part of the swap.

To traverse the entire neighbourhood all pairs of orders must be examined (skipping pairs where both orders are assigned to the same row), and the size of the neighbourhood is therefore $O(n^2)$.

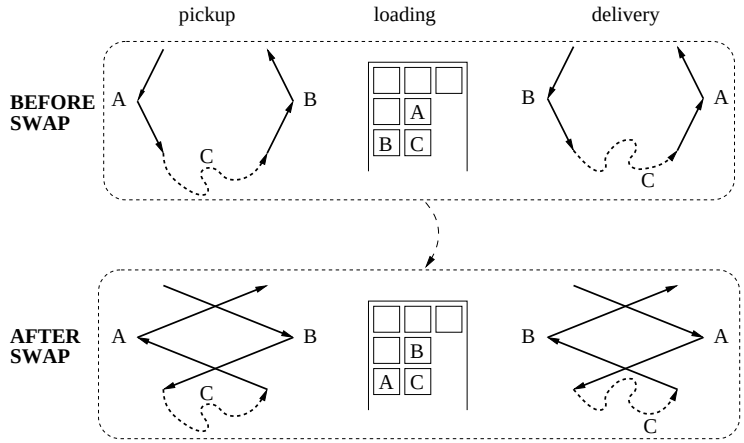


Figure A.2: An illustration of *complete-swap*.

A.3.2.1 The operators in combination.

Since *route-swap* does not affect the row assignment of a solution, it is obvious that one cannot reach the entire solution space by only using this operator. Similarly when performing *complete-swap* the mutual visiting orders of the two routes will never change (e.g. if an order is handled third in the pickup route and last in the delivery route, then there can never be an order visited third in the pickup route and not last in the delivery route, if only *complete-swap* is applied). However by using a combination of the two operators it becomes possible to cover the entire solution space.

The idea of combining several different neighbourhood structures is not unique – it is also the idea behind Variable Neighbourhood Search (VNS, see [80]). However, the background for using this approach here is different. The purpose of VNS is to be able to reduce solution time, while the use of a combined neighbourhood in this paper is a necessity to ensure access to all feasible solutions.

Any solution can be constructed from any other solution by first performing a series of *complete-swaps* until the loading plan is correct, followed by a number of *route-swap* operations to make the routes match. When performing a *route-swap* only one of the routes will be affected, unless the swapped items are assigned to the same row. In this case the two items *must* be swapped in both routes as the solution would otherwise become infeasible.

A.3.3 Iterated Local Search

The idea behind iterated local search (cf. [76]) is to use a simple local search procedure a number of times in an intelligent way, to eventually produce good results. In the current implementation this has been done

by using steepest descent at the local search procedure, and using random restarts as a rather crude iteration control. The restart mechanism is usually at the heart of the algorithm, and thus the implementation here should not be seen as a representative of what ILS would be capable of for this problem. However it was implemented to examine the value of using a pure steepest descent approach, and the results should be judged as such.

A.3.4 Tabu Search

In the tabu search implementation the fact that two different neighbourhood structures must be considered has been dealt with by systematically changing between the two types, following some pre-defined pattern. Two parameters are used to describe this pattern, namely the *length* or *period* of the pattern, `perLen`, and the *ratio* between the two operators, `ratio`. E.g. the parameter combination $\texttt{perLen} = 20, \texttt{ratio} = 0.3$ indicates that the first 6 iterations should use *route-swap*, the next 14 should use *complete-swap*, and from iteration 21 onwards this pattern is repeated.

In this way the deterministic nature of tabu search has been maintained, by not introducing randomness in the selection of the operator to use for each iteration.

For each operator it has additionally been necessary to decide on some attributes to register in the tabu list, and for both operators the choice has been to register which two orders were swapped.

For moves of type *route-swap* it is thus *not* registered on which route (pickup or delivery) the swap was performed.

Although moves of both types are marked by the same attributes in the tabu list, a separate list is kept for each move type. This is due to the differences in the two operators. Consecutively performing one move of each type on the same pair of orders does not imply that one move is reversing the other, and does not lead the algorithm to start cycling between a few solutions, and hence the two operators should not be kept in the same tabu list.

The TS algorithm implemented here is rather simple, with a constant tabu list length, fixed switching strategies between the operators, and a stopping criterion based on elapsed time. Using a different stopping criterion, such as a certain number of iterations without improvement, was decided against to ensure comparability between all of the implemented algorithms.

A.3.5 Simulated Annealing

The SA implementation handles the two available operators by randomly selecting one of them for each iteration, and the probability for choosing each operator is expressed in the parameter *ratio*, indicating the probability of choosing *route-swap*.

Traditionally SA approaches take three parameters; an initial temperature T_s, a final temperature T_e, and a temperature reduction function.

The temperature reduction function is usually based on the reduction scheme

$$T_{i+1} = c \cdot T_i \tag{A.11}$$

where $T_0 = T_s$, $c \in (0; 1)$ is some reduction factor, and the algorithm terminates when the final temperature T_e is reached.

However for the purpose of comparability SA has here been implemented to take running time t as an input parameter instead of the temperature reduction factor/function.

Thus the temperature at time t has been calculated as:

$$T(t) = T_s \cdot \left(\frac{T_e}{T_s}\right)^{\frac{t}{t_{\max}}} \tag{A.12}$$

where $t_{\max}$ is the total allowed running time. This ensures that $T(0) = T_s$, $T(t_{\max}) = T_e$ and the behaviour between these two points corresponds to that of (A.11) when temperature T_e is reached at time $t_{\max}$. In this implementation the temperature is updated at each iteration.

A.3.6 Large Neighbourhood Search

Large neighbourhood search (cf. [104]), is based on simply removing (larger) parts of the solution, and subsequently reinserting the affected customers. Thus it does not use neighbourhood operators in the same sense as the previously discussed local search based metaheuristics.

An implementation of LNS must consider such issues as strategies for removal and insertion, degree of destruction at each iteration and acceptance criterion for the generated solutions.

In this case both the removal and insertion strategies are based on a variety of simple strategies, where the strategy to apply at each operation is selected randomly with some probability (which is a parameter). As with the other algorithms used in this paper, the stopping criterion for LNS has been running time.

A.3.6.1 Operators

Two removal strategies have been implemented – one is based on a measure of *relatedness* similarly to the description in [104], thus attempting

to remove orders similar to those already removed in the same iteration, while the other is based on removing the orders that are most expensive to cover in the current solution. In this context the relatedness of two orders has been expressed as the sum of the distances between the orders in each of the two graphs.

Reinsertion is performed using a range of four different insertion heuristics: Nearest, farthest, cheapest and most expensive (all based on [63]).

Additionally, a certain amount of noise has been added when choosing the order to insert, to increase diversification.

A.3.6.2 Parameters

In addition to the use of the available operators a number of parameters have been applied and tested for the LNS implementation.

The criterion for accepting new solutions has been based on simulated annealing acceptance, similar to the approach used in [96]. As in Section A.3.5 the calculation of the temperature is based on a pre-determined stopping criterion, rather than using the temperature itself as the stopping criterion.

Additionally it has been decided to remove *orders* from the solution rather than *customers*. This is based on the structure of the DTSPMS, which strongly reduces the reinsertion possibilities, if e.g. a pickup customer has been removed, while the corresponding delivery customer remains in the solution.

Finally, the steepest descent used for ILS has been used to improve the solution after each iteration.

A.4 Computational Results

Each of the four algorithms has been implemented in Java 1.5 and tests have been performed for running times 10 and 180 seconds (both are wall clock times).

All tests have been performed on a Dell D610 laptop with 1.5 GB RAM and a 1.60 GHz processor running Windows XP.

A.4.1 Test Instances

The test instances that have been used for the evaluation of the solution approaches have been generated randomly, by finding two sets of n random (real) points in a 100×100 square. The depot is placed in the centre of the square at $(50, 50)$. All distances are Euclidean distances rounded to the nearest integer, in accordance with the conventions from TSPLIB. Note that this rounding implies that the triangle inequality is

not preserved, however the algorithms described in this paper do not rely on this inequality to be valid.

Two test sets have been generated, each containing ten problems of size $n = 33$ orders (i.e. with 33 customers in each graph). The first set was used for parameter tuning for each of the heuristics, and both sets were used for the final results reported in Table A.2.

The test instances generated for this problem can be obtained online[1].

In this paper all test problems have been solved for the 3-stack problem.

A.4.2 Bounds

Since the optimal solutions to the test instances are unknown, it is desirable to find good lower bounds to evaluate the quality of the solutions obtained during testing.

One lower bound can be found by relaxing the constraints that ensure loading feasibility between the graphs, i.e. solving the n-stack problem. This way the problem reduces to two individual TSPs, that can be solved to optimality with tools such as Concorde (cf. [2]). Obviously each TSP solution is a lower bound on the optimal tour through that graph, and the sum of the tours is then a lower bound on the sum of the tours in any feasible solution to the DTSPMS with any number of rows.

However, intuitively this bound must be expected to be quite weak, since one would expect many changes to be necessary before a feasible loading plan could be constructed for such a solution.

In addition to this problem-specific lower bound some common lower bounds have been calculated and compared, to demonstrate some properties of the problem. These numbers can be found in table A.1.

The first column gives the problem number. The second column (nS) gives the lower bound described above (the optimal solution to the n-stack problem), column LP gives the lower bound provided by solving the LP relaxation of the model (A.1)–(A.10) and root gives the objective value in the root node after CPLEX has added cuts. BB shows the lower bound obtained by CPLEX after an hour of running time with BestBound used as the node selection strategy. Best shows the value of the best known solution for each problem, while $\frac{\text{Best}}{\text{LB}}$ shows the quality of the best known solution compared to the best lower bound (i.e. $\frac{\text{Best}}{n\text{S}}$). All best known solutions have been found by allowing the best metaheuristic (LNS) to use a longer running time. The last two columns contain upper bounds: SS gives the optimal solution to the single-stack problem, while the column *init* gives the heuristic solution to problem SS, which was used as the initial solution that was used for TS, SA, and LNS.

As can be seen from the table the n-stack bound is always considerably

[1] `http://www.transport.dtu.dk/datasets/DTSPMS`

	nS	LP	root	BB	Best	$\frac{\text{Best}}{\text{LB}}$	init	SS
R00	911	793	802	812	1063	1.17	1886	1682
R01	875	820	824	831	1032	1.18	1690	1579
R02	935	826	844	845	1065	1.14	1672	1564
R03	961	887	905	905	1100	1.14	1836	1741
R04	937	859	860	863	1052	1.12	1671	1629
R05	900	811	814	816	1008	1.12	1548	1438
R06	998	941	944	944	1110	1.11	1739	1643
R07	963	894	900	900	1105	1.15	1867	1696
R08	978	899	911	922	1109	1.13	1761	1643
R09	976	889	909	910	1091	1.12	1610	1556
R10	901	822	833	839	1016	1.13	1697	1575
R11	892	810	820	823	1001	1.12	1494	1429
R12	984	934	946	950	1109	1.13	1778	1673
R13	956	887	895	897	1084	1.13	1707	1613
R14	879	794	803	803	1034	1.18	1704	1565
R15	985	903	916	917	1142	1.16	1943	1783
R16	967	857	887	894	1093	1.13	1767	1647
R17	946	847	882	884	1073	1.13	1716	1620
R18	1008	876	920	921	1118	1.11	1796	1673
R19	938	839	855	864	1089	1.16	1725	1633

Table A.1: Bounds.

stronger than the LP bound, even after CPLEX has been allowed some time for improving this bound. Comparisons show that the best known solutions are all within 11–18% of the best lower bound (nS). Given that this lower bound is found by completely ignoring all sequencing constraints, one may assume that these lower bounds are quite weak. This suggests that the best found solutions could be reasonably good.

The best known solutions have all been found by performing a number of runs of length around two hours. The best approach for obtaining good solutions has turned out to be running several runs of a somewhat shorter length, rather than performing one run with a very long running time. Performing one run of 15 minutes length gives solutions that are on average 0.5% above the best known, and when performing four runs of each 90 minutes (i.e. 6 hours total time), the best known solution was matched at least once for all 20 instances.

A.4.3 Results

A number of test runs have been performed for each of the four algorithms implemented. The details of these implementations will first be presented individually before presenting the final results and comparisons in Table A.2. It is worth noting that the first three implementations all use the two improvement operators that were designed specifically for the DTSPMS

and presented in Section A.3.2, while the LNS implementation instead uses more general removal and insertion operators as described in Section A.3.6.1.

All of the algorithms required considerations regarding the ratio between the different operators, and apart from the ILS algorithm each includes a number of parameters that additionally needed to be calibrated.

All calibration runs were performed on the problems from the first data set to determine a good set of parameters, and these parameters were then applied to the problems from the both of the two data sets to produce the final results presented in Table A.2. More elaborate results of the calibration runs can be obtained from [90], or by contacting the author.

In the remainder of this paper the term *solution quality* will be used to refer to the deviation from the best known solution, i.e. the objective value of the solution to an instance is divided by the value of the best known solution for that instance.

A.4.3.1 Iterated Local Search

For each iteration of the local search algorithm the operator to use has been chosen randomly with some probability *ratio*. The best results were obtained with value 0.4 for both of the running times considered, meaning that 40% of the moves were of type *route-swap*.

The algorithm could typically complete around 2600 iterations within the allotted 3 minutes.

A.4.3.2 Tabu Search

Next the impact of the length of the tabu list, *tabuLength*, was examined. Based on the results, tabu lengths of 7 and 11 were used for the remaining runs, for 10 and 180 seconds, respectively.

Finally, the combination of the operators was examined, given by the parameters *ratio* and *perLen* as described in Section A.3.4. It proved beneficial to use a low value for *perLen* (tested values: 10, 30, 50, 70, 90), namely 10 for both running times. It was assessed that 10 was so close to 0 that using an even lower value would not produce any significant gain. The best value of *ratio* turned out to be higher for tabu search than for iterated local search, with values of 0.9 for 10 seconds and 0.7 for 180 seconds runs, thus using the less time-consuming operator more frequently, especially for the shorter TS runs (tested values: 0.1, 0.3, 0.5, 0.7, 0.9).

The algorithm was able to complete around 137,000 iterations within the allotted 180 seconds (ratio 0.7).

This strong preference for using the faster operator also gives an indication, that the number of iterations that can be completed is important to the result (number of iterations completed is approximately 54,000 for ratio 0.1 and 88,000 for ratio 0.5). This in turn indicates that the use of one combined neighbourhood might not pay off, since the number of iterations would then be reduced by 50%.

A.4.3.3 Simulated Annealing

The *ratio* parameter turned out to have a rather small impact on the quality of the solutions obtained by simulated annealing, and the chosen values were 0.4 for 10 seconds and 0.5 for 180 seconds runs. These tests were furthermore performed with ratios 0.0 and 1.0, to demonstrate the effect of using only one operator. As expected this approach produced considerably worse solutions, and for running time 180 seconds the solution quality was 1.47 (*route-swap* only) and 1.35 (*complete-swap* only) when using one operator exclusively, while the corresponding numbers were around 1.1 for all tested values of *ratio* in the interval $[0.1; 0.9]$.

Finally, several combinations of start and end temperatures were examined, to determine the final set of parameters.

With these parameters the algorithm could complete around 15 million iterations in 3 minutes.

A.4.3.4 Large Neighbourhood Search

For both running times it showed beneficial to use a relatedness-based removal strategy in most iterations, with some uses of the most-expensive strategy for increased diversification. The final setting used relatedness in 80% of the iterations for 180 seconds runs, and 60% for 10 seconds runs.

The best strategy for insertion in all cases was to randomly select one of the four available insertion strategies.

The number of orders to remove was selected randomly at each iteration – in the interval [4, 17] for 3 minutes and [3, 15] for 10 seconds. This indicates that the longer runs could benefit from a higher degree of diversification than the shorter runs.

Additionally it was tested to use the steepest descent algorithm from the ILS for re-optimisation at each iteration. The longer 3 minute running times would benefit from this refinement, while the shorter 10 second runs performed better when using less refinement and spending the time on additional iterations.

The acceptance of new solutions was performed using the criterion from simulated annealing – there was no variation in temperature interval between the two running times.

Finally an amount of noise was added to the solution when determining which order to reinsert next. Here the 10 second runs showed an overall best performance when these values were modified with ±40%, while the corresponding number for 3 minutes was ±20%.

The LNS algorithm could complete around 12,500 iterations within 3 minutes.

A.4.3.5 Summary of the Results

After calibrating each of the four heuristics, Table A.2 summarises the results obtained by applying these parameters to the test problems from both sets.

	10 seconds				180 seconds			
	LNS	SA	TS	ILS	LNS	SA	TS	ILS
R00	1.04	1.26	1.42	1.66	1.01	1.13	1.23	1.58
R01	1.04	1.17	1.34	1.64	1.01	1.08	1.21	1.61
R02	1.04	1.19	1.38	1.57	1.01	1.10	1.24	1.53
R03	1.06	1.22	1.44	1.66	1.01	1.12	1.24	1.62
R04	1.05	1.25	1.38	1.59	1.02	1.11	1.23	1.56
R05	1.03	1.22	1.25	1.53	1.01	1.15	1.21	1.47
R06	1.06	1.19	1.37	1.56	1.02	1.11	1.26	1.51
R07	1.05	1.23	1.39	1.66	1.01	1.11	1.26	1.58
R08	1.04	1.21	1.36	1.56	1.01	1.11	1.27	1.51
R09	1.04	1.15	1.29	1.47	1.01	1.08	1.19	1.46
R10	1.05	1.24	1.45	1.67	1.00	1.15	1.25	1.64
R11	1.06	1.24	1.24	1.49	1.01	1.10	1.22	1.48
R12	1.04	1.21	1.44	1.60	1.01	1.13	1.22	1.55
R13	1.04	1.23	1.37	1.55	1.01	1.08	1.22	1.53
R14	1.03	1.22	1.47	1.63	1.00	1.11	1.25	1.58
R15	1.04	1.21	1.37	1.62	1.01	1.10	1.26	1.56
R16	1.02	1.20	1.35	1.61	1.00	1.10	1.18	1.55
R17	1.04	1.24	1.48	1.60	1.00	1.12	1.28	1.58
R18	1.05	1.20	1.33	1.59	1.01	1.13	1.21	1.53
R19	1.03	1.16	1.31	1.57	1.01	1.11	1.25	1.54
Avg. set 0	1.04	1.21	1.36	1.59	1.01	1.11	1.23	1.54
Avg. set 1	1.04	1.22	1.38	1.59	1.01	1.11	1.23	1.56

Table A.2: Result Summary.

The table shows the quality of the solutions that have been found with each of the four heuristic approaches for the two different running times. All values are found using the "best" set of parameters found in the preceding sections.

The table shows that among the four implementations presented, LNS consistently produces the best results, and gives results around 4% above the best known for a running time of 10 seconds, and within 1–2% when

allowed to run for 3 minutes.

Additionally the table shows a very clear ranking of the implementations. For all but the ILS a significant improvement can be observed by increasing the running time.

The results also show that using the ILS algorithm with the short running time produces solutions that are only slightly better than the initial solutions used for the two other approaches. This demonstrates that a clear benefit can be obtained by using something more sophisticated than the current ILS, which is basically a series of steepest descents with random restart.

The top half of the table (problems R00–R09) shows the results obtained for set 0 which was used for parameter tuning, while the bottom half (problems R10–R19) shows the results for set 1 which has been used exclusively for the final evaluation of the algorithms. Averages for each set are reported at the bottom of the table. This shows that the solution quality is not significantly higher for the problems that have been used for calibration, i.e. the choice of calibration problems does not seem to have influenced the final parameter values.

A.4.4 Instances with known optimal solution

The largest instance size which can currently be solved to optimality with the mathematical model presented in Section A.2 contains 12 customers in each graph. A series of such instances has been obtained by considering the first 12 orders of each of the above-mentioned instances. Table A.3 gives the optimal solution to each of these reduced instances, along with the average gap obtained by running the LNS, SA, and TS algorithms on these. The gap is averaged over three runs for LNS and SA, and in parentheses is given the number of times (out of three) where the optimal solution was found.

The first column of Table A.3 shows the optimal value, and the next two columns show the quality of the lower bounds obtained from the n-stack problem, and by letting CPLEX add cuts to the root node (cf. the bounds presented in Section A.4.2). Solution times for obtaining the optimal solution range from 14 to 2850 seconds, with an average of 450.

Again the nS bound turns out to be superior in all cases, and in this case provides a quite good bound – probably due to the size of the problems. It should still be expected that the quality of this bound will deteriorate with increased values of L, since the nS bound is exactly the optimal solution for $R = n$. A few tests that could be completed with $R = 3, L = 5$ confirm this expectation.

These results confirm the superiority of the LNS implementation over the three other implementations, by consistently finding the optimal solution, and also confirm that SA performs better than TS.

					10 seconds			180 seconds		
	opt	$\frac{n\text{S}}{\text{opt}}$	$\frac{\text{root}}{\text{opt}}$	$\frac{\text{LP}}{\text{opt}}$	LNS	SA	TS	LNS	SA	TS
R00-12	694	0.98	0.96	0.96	1.00 (3)	1.02 (0)	1.04 (0)	1.00 (3)	1.01 (1)	1.04 (0)
R01-12	710	1.00	0.89	0.86	1.00 (3)	1.03 (0)	1.06 (0)	1.00 (3)	1.02 (0)	1.06 (0)
R02-12	606	0.98	0.91	0.89	1.00 (3)	1.02 (0)	1.08 (0)	1.00 (3)	1.01 (0)	1.08 (0)
R03-12	680	0.99	0.96	0.96	1.00 (3)	1.01 (0)	1.06 (0)	1.00 (3)	1.00 (3)	1.06 (0)
R04-12	607	0.99	0.95	0.93	1.00 (3)	1.03 (0)	1.09 (0)	1.00 (3)	1.00 (3)	1.06 (0)
R05-12	567	0.99	0.89	0.87	1.00 (3)	1.00 (2)	1.12 (0)	1.00 (3)	1.00 (3)	1.13 (0)
R06-12	747	0.99	0.95	0.90	1.00 (3)	1.03 (1)	1.08 (0)	1.00 (3)	1.00 (2)	1.08 (0)
R07-12	557	0.96	0.90	0.85	1.00 (3)	1.02 (0)	1.03 (0)	1.00 (3)	1.00 (3)	1.03 (0)
R08-12	690	0.98	0.98	0.98	1.00 (3)	1.04 (0)	1.13 (0)	1.00 (3)	1.00 (3)	1.12 (0)
R09-12	669	1.00	0.97	0.92	1.00 (3)	1.02 (0)	1.08 (0)	1.00 (3)	1.00 (1)	1.07 (0)
R10-12	633	0.95	0.93	0.91	1.00 (3)	1.01 (0)	1.11 (0)	1.00 (3)	1.00 (2)	1.08 (0)
R11-12	591	0.96	0.95	0.93	1.00 (3)	1.04 (0)	1.07 (0)	1.00 (3)	1.00 (3)	1.07 (0)
R12-12	722	0.99	0.92	0.91	1.00 (3)	1.01 (0)	1.09 (0)	1.00 (3)	1.00 (2)	1.08 (0)
R13-12	664	0.97	0.96	0.95	1.00 (3)	1.02 (2)	1.09 (0)	1.00 (3)	1.00 (3)	1.10 (0)
R14-12	650	0.98	0.85	0.84	1.00 (3)	1.03 (0)	1.07 (0)	1.00 (3)	1.01 (2)	1.07 (0)
R15-12	595	0.97	0.95	0.93	1.00 (3)	1.01 (2)	1.01 (0)	1.00 (3)	1.00 (2)	1.03 (0)
R16-12	577	0.99	0.98	0.98	1.00 (3)	1.02 (0)	1.05 (0)	1.00 (3)	1.00 (3)	1.05 (0)
R17-12	737	0.99	0.97	0.95	1.00 (3)	1.01 (1)	1.07 (0)	1.00 (3)	1.00 (1)	1.07 (0)
R18-12	724	0.98	0.98	0.97	1.00 (3)	1.01 (0)	1.07 (0)	1.00 (3)	1.00 (2)	1.07 (0)
R19-12	753	0.99	0.95	0.92	1.00 (3)	1.02 (0)	1.11 (0)	1.00 (3)	1.00 (2)	1.11 (0)
Avg.		0.98	0.94	0.92	1.00 ($\frac{60}{60}$)	1.02 ($\frac{8}{60}$)	1.08 (0)	1.00 ($\frac{60}{60}$)	1.00 ($\frac{41}{60}$)	1.07 (0)

Table A.3: Results for 12 order instances.

A.4.5 Larger problem instances

A real-life application containing 66 orders can also be imagined if the transported items are all vertically stackable and have a height that is less than half of the height of the container. In this case the top item of such a vertical stack must obviously be removed first, and additionally all items from one position must be removed before the top item on the next position becomes available. Thus solving an instance using such stacking of height 2 is identical to solving an instance with rows of double length.

The previously used test instances have all been increased to size 66 by letting the original random generation procedure continue until the desired number of orders has been generated (these larger instances are also available from the same website), and a series of test runs have been performed on these instances with the parameter settings found earlier. The results hereof can be found in Table A.4.

			10 seconds			180 seconds		
	Best	nS	LNS	SA	TS	LNS	SA	TS
R00-66	1594	1237	1.19	1.38	1.59	1.07	1.21	1.45
R01-66	1600	1257	1.20	1.39	1.60	1.08	1.22	1.57
R02-66	1576	1295	1.20	1.38	1.63	1.12	1.28	1.61
R03-66	1631	1290	1.14	1.35	1.58	1.06	1.23	1.53
R04-66	1611	1295	1.18	1.46	1.52	1.09	1.28	1.43
R05-66	1528	1204	1.18	1.40	1.53	1.07	1.21	1.49
R06-66	1651	1294	1.17	1.38	1.56	1.07	1.21	1.51
R07-66	1653	1307	1.17	1.44	1.56	1.08	1.22	1.41
R08-66	1607	1297	1.18	1.41	1.54	1.07	1.27	1.52
R09-66	1598	1276	1.18	1.35	1.59	1.08	1.25	1.53
R10-66	1702	1339	1.17	1.35	1.60	1.09	1.23	1.49
R11-66	1575	1268	1.19	1.40	1.55	1.08	1.25	1.49
R12-66	1652	1295	1.19	1.38	1.55	1.10	1.24	1.41
R13-66	1617	1275	1.19	1.39	1.60	1.10	1.24	1.53
R14-66	1611	1245	1.21	1.38	1.65	1.09	1.24	1.54
R15-66	1608	1228	1.19	1.38	1.56	1.10	1.22	1.48
R16-66	1725	1356	1.16	1.34	1.51	1.07	1.17	1.41
R17-66	1627	1274	1.21	1.40	1.70	1.10	1.29	1.59
R18-66	1671	1328	1.18	1.40	1.58	1.08	1.23	1.52
R19-66	1635	1256	1.17	1.41	1.57	1.09	1.22	1.48
Avg.			1.18	1.39	1.58	1.08	1.24	1.50

Table A.4: Results for 66 order instances.

Again it can be seen that LNS outperforms the other heuristics.

Additionally it can be noticed that the gap between the lower bound and the best known solution has increased notably (avg. 30%). This probably indicates that these instances are harder and the best known solution could therefore be further away from the optimum, but most likely also reflects that when the number of items in each loading row

increases, the quality of the n-stack lower bound decreases.

The best known solutions to these instances have again been found by performing a series of runs of a few hours, however a lot more runs were required for these instances than for the smaller instances of 33 customers. Performing two runs of 4 hours each (thus 8 hours total time), will generate a solution that is around 1% above the best known, which reduces to 0.53% above the best known for 8 times 4 hours (i.e. 32 hours total), to 0.35% above for 16 times 4 hours, and to 0.25% for 25 times 4 hours (total 100 hours). Different combinations of running time/number of runs have been tested, and a series for 4 hour runs generally gave the best results.

A.5 Conclusion and future work

This paper has introduced the DTSPMS which is a new variant of the TSP/PDP, presented a mathematical formulation of the problem, and demonstrated the behaviour of different heuristic solution approaches on the problem.

Four different implementations have been tested, and comparisons have shown that large neighbourhood search produces the best results of the four, with solutions that are within 2% of the best known solution, with a running time of 3 minutes, and within 2–6% for running times of 10 seconds.

It seems clear that the strongly restricted nature of this problem should be taken into consideration when choosing a suitable solution approach. The results obtained in this paper indicate that the tested traditional metaheuristic local search solution approaches might be insufficient for producing good solutions to the problem – apparently they are unable to cover the irregular solution space well enough. Instead the LNS-algorithm, which allows free movement through the solution space, proves successful. This observation is also supported by the preliminary results of [34], which uses intermediate partial solutions and larger neighbourhoods than the ones presented in the three initial heuristics of this paper.

Additionally, tests have been performed on smaller instances with known optimal solutions, and on larger instances to further examine the behaviour of the implemented algorithms, and the findings are consistent with the initial results.

A.5.1 Future work

Future work on the DTSPMS could focus on either improving the solutions/solution methods or generalising the problem.

One way of attempting to improve the solutions would be by improving

the approaches already described here. In particular one could attempt refining the tabu search, e.g. by using variable tabu list length, or by introducing some diversification mechanism, such as performing mutations to the solution when a certain number of iterations have been performed without leading to any improving solutions. Furthermore using a randomised operator choice for tabu search might lead to improvement, since the shorter period lengths are consistently the best. However, although some improvement of the behaviour of the three initial metaheuristics could be obtained, it is questionable whether the improvement would match the results of the LNS.

The three initial metaheuristics are all based on using a combination of different neighbourhoods, and to this end it would also be possible to consider different switching strategies, other than fixed ratios and random selection at each iteration. This could be obtained by using only one operator at a time until a certain number of iterations has been unable to produce improving solutions, or by using some adaptive scheme where the selection probability depends on previous successes.

The current ILS implementation was not intended as an ILS as such, but simply an attempt to test the usefulness of using a very simple steepest descent approach repeatedly. Thus the implementation could be refined using the usual principles from ILS, in particular improving the restart procedure to start at a perturbation of a previous solution, rather than a completely random solution at each iteration.

Additionally it could be attempted to solve the DTSPMS to optimality, and it could be interesting to examine the effect of the number of loading rows on the solution.

The most obvious extension of the problem is to either generalise the TSP aspects of the problem to more general vehicle routing (VRP) and include multiple vehicles/containers and/or multiple depots to form a Double VRP with Multiple Stacks, or to generalise the problem in the direction of the regular pickup and delivery problem, to form a pickup and delivery problem with multiple stacks (PDPMS).

In the first case, if the choice of depot becomes a decision, it is likely that the cost of the long-haul between the depots will no longer be negligible, and this may be very hard to estimate at the time of planning in real-life applications.

Modifying the problem to include non-uniform objects would complicate the packing considerably, since some objects might still be positioned next to each other, so that their internal delivery order becomes irrelevant. This would heavily influence the use of the LIFO-principle so far, and would approach the problem to the more generalised PDP with loading constraints.

Appendix B

Exact Solutions to the Double Travelling Salesman Problem with Multiple Stacks

Authors:
Hanne L. Petersen, Claudia Archetti, M. Grazia Speranza.

Abstract:
In this paper we present mathematical programming formulations and solution approaches for the optimal solution of the Double Travelling Salesman Problem with Multiple Stacks (DTSPMS). A set of orders is given, each one requiring transportation of one item from a customer in a pickup region to a customer in a delivery region. The vehicle available for the transportation in each region carries a container. The container is organised in rows of given length. Each row is handled independently from the others according to a LIFO (Last In First Out) stack policy. The DTSPMS problem consists of determining the pickup tour, the loading plan of the container and the delivery tour in such a way that the total length of the two tours is minimised. The formulations are based on different modelling ideas and each formulation gives rise to a specific solution approach. We present computational results on a set of benchmark instances that compare the different approaches and show that the most successful one is a decomposition approach applied to a new model.

B.1 Introduction

The Double Travelling Salesman Problem with Multiple Stacks (DTSPMS) was first introduced in [91], which presented a mathematical formulation and some metaheuristic solution approaches to the problem. In this paper we face the problem of solving the DTSPMS exactly.

In the DTSPMS a set of orders is given, each one requiring transportation of one item from a customer in a pickup region to a customer in a delivery region. The two regions are separated from each other and are typically far apart. The vehicle available for the transportation carries a container. The vehicle starts from the depot of the pickup region, is loaded during the pickup phase, transported from the depot of the pickup region to the depot of the delivery region and then unloaded in the delivery region. The container is organised in rows of given length. All items are boxes/pallets of identical dimension and each item occupies one of the positions of a row. Each row is independently handled from the others according to a LIFO (Last In First Out) stack policy as items are loaded and unloaded from the back of the vehicle. No repacking is allowed at any time, not even between the pickup and the delivery phases. The problem consists of determining the pickup tour, the loading plan of the container and the delivery tour in such a way that the total length of the two tours is minimised. The interest in this problem comes from a real world application (see [91]). The problem finds applications in pickup and delivery freight transportation and in multi-modal transportation, where the pickup operations are followed by the air or railroad transportation of the container to the delivery region.

The DTSPMS is a variation of the Travelling Salesman Problem with Pickup and Delivery (TSPPD) and shows some similarities to the TSPPD with LIFO loading (TSPPDL). In comparison to the latter the DTSPMS has its pickup and delivery operations completely separated, which provides a simplification, and introduces the concept of several loading rows, which increases the complexity.

The most closely related earlier work on exact solutions to the TSPPDL has been done by Cordeau et al. [23], who present three different models for the TSPPDL and a branch-and-cut approach. The two first models use continuous variables to represent the load of the vehicle, while the last model solves the problem without adding extra variables, and instead imposes additional constraints on subsets of the existing variables. Additionally, the TSPPDL is treated using exact methods by Carrabs et al. [11], who uses a branch-and-bound approach based on additive lower bounds, and heuristically by Carrabs et al. [12], using variable neighborhood search with a collection of old and new operators. A closely related problem is also the TSPPD with FIFO (First In First Out) loading studied in [11, 22, 33].

A more extensive literature exists for different variations of the vehi-

cle routing problem (VRP) with loading constraints, which have less in common with the DTSPMS, although the DTSPMS is a special case of a routing problem with 2-dimensional loading constraints and identically shaped objects. In [61], Iori et al. solve the capacitated VRP with respect to some two-dimensional loading constraints, using an infeasible path approach that is somewhat related to the one used in this paper. The authors use a branch-and-cut approach which first solves the problem with all loading constraints relaxed, and subsequently adds infeasible paths based on loading infeasibilities.

In this paper, the mathematical programming formulation of the DTSPMS introduced in [91], which is here called the precedence model, is used in a branch-and-cut approach. One variation of this model is proposed together with two new different mathematical formulations (the flow model and the Travelling Salesman Problem with Infeasible Paths model) in the hope of finding a formulation that could lead to the exact solution of instances of larger size. The Travelling Salesman Problem with Infeasible Paths model has been solved through a decomposition approach that turned out to be effective. The tests that compare the different formulations confirm that this problem is extremely hard, as is the case for the other above mentioned problems that combine routing and loading.

In this paper we first give a description of the problem in Section B.2 followed by Sections B.3, B.4 and B.5 which present the precedence model and its variation, the flow model and the Travelling Salesman Problem with Infeasible Paths model, respectively, together with the solution methods. Finally, Section B.6 presents computational results, and some concluding remarks are given in Section B.7.

B.2 Problem description

The Double Travelling Salesman Problem with Multiple Stacks (DTSPMS) consists of finding the shortest routes performing pickups and deliveries in two separated regions.

A set of *pickup* customers have to be served with a pickup operation of an item. A *delivery* customer, to whom the item has to be delivered, is associated with each pickup customer. All the pickup operations are carried out before any delivery operation can take place. The available vehicle carries a container that is organised in *rows* (horizontal stacks) of a given length. After all the pickup operations have taken place, the container is moved to the delivery region and the delivery phase begins. When picked up, an item is inserted into the first available position of a specific row and its position cannot be changed later. The row in which to insert it is to be decided. For example, if there are three rows of length 5 and 3 items have already been picked up and inserted into row 1, the next item that is inserted in row 1 will occupy position 4. One

more position then remains available in row 1. If space is available in other rows a crucial decision consists of choosing the row for inserting the item. This decision is crucial because each row is served according to a LIFO (Last-In-First-Out) policy in the delivery phase, as the items are loaded and unloaded from the back of the vehicle. At any time of the delivery phase the last inserted, and not yet unloaded, item of each row can be unloaded. Neither repacking nor vertical stacking is permitted at any time.

A number of parameters are used to describe the configuration of a given problem instance, namely the number of orders (and thereby items) N, the number of rows, R, and the length of the rows, L. Each order identifies a pickup and a delivery customer. Thus, the total number of graph nodes (customers and depots) visited by the vehicle is $n = 2N+2$. In most cases, and unless otherwise stated, we assume $N = R \times L$, i.e., the full loading capacity is used.

We denote each order (and thereby item) with the index i. Let $\mathcal{I}$ be the set of orders $\mathcal{I} = \{1, \ldots, N\}$. The problem is defined on two complete, oriented and unconnected graphs: the graph of the pickup customers for which we use the letter P, and the graph of the delivery customers for which we use the letter D, with $\mathcal{T} = \{\mathrm{P}, \mathrm{D}\}$. We denote either of the two graphs by $G^T = (V^T, A^T)$, $T \in \mathcal{T}$. In order to avoid excess notation and with a slight abuse of notation, the index i, that is used to indicate an order (and the associated item to be picked up and delivered), will be also used to indicate the node of graph G^{P} associated with the pickup customer and the node of the graph G^{D} associated with the delivery customer. Thus, an item i is picked up at node i in G^{P} and delivered to node i in G^{D}. Graph G^T has $N+1$ nodes, $V^T = \mathcal{I} \cup \{0\}$, with nodes $i = 1, \ldots, N$ representing customers and node 0 representing the depot. A cost c_{ij}^T is associated with each arc $(i,j) \in A^T, T \in \mathcal{T}$.

Each loading row has a limited capacity, meaning that the number of items kept in a row cannot exceed a certain value. As a single vehicle is considered throughout the paper, the number of items to be transported will not exceed the total capacity of the rows of the vehicle. We will denote by $\mathcal{R}$ the set of rows and will write $r \in \mathcal{R}$ to indicate that the row r belongs to the set $\mathcal{R}$.

It can be noted that if both the pickup and delivery route of a feasible solution to the DTSPMS are reversed and the assignment of the items to the rows is maintained, we obtain a feasible solution (with the ordering of each row reversed). If the distance matrix is symmetric the objective value will be unchanged.

Cordeau et al. [23] make a comparison of the number of feasible solutions to the TSPPDL depending on the number of orders. The corresponding numbers for the DTSPMS have been calculated as described in the Appendix and a comparison of the values can be found in Table B.1. The values for the TSPPDL for $N > 8$ have been found by using the formula

given in [23]. We denote by DTSPxS the DTSPMS with x rows. All values have been calculated considering the shortest row length possible, $L = \lceil \frac{N}{R} \rceil$. The values for DTSP$x$S in italics indicate non-full loadings ($\frac{N}{R}$ not integer).

N	TSPPDL	DTSP2S	DTSP3S
4	336	864	*69984*
5	5040	*48000*	*972000*
6	95040	288000	$5.83 \cdot 10^6$
7	$2.16 \cdot 10^6$	*$2.47 \cdot 10^7$*	*$5.55 \cdot 10^9$*
8	$5.77 \cdot 10^7$	$1.98 \cdot 10^8$	*$1.14 \cdot 10^{11}$*
9	$1.76 \cdot 10^9$	*$2.30 \cdot 10^{10}$*	$1.02 \cdot 10^{12}$
10	$6.09 \cdot 10^{10}$	$2.30 \cdot 10^{11}$	*$1.76 \cdot 10^{15}$*
11	$2.35 \cdot 10^{12}$	*$3.41 \cdot 10^{14}$*	*$4.79 \cdot 10^{16}$*
12	$9.96 \cdot 10^{13}$	$4.09 \cdot 10^{14}$	$5.75 \cdot 10^{17}$

Table B.1: Number of feasible solutions with N orders.

B.2.1 Remarks

When a problem instance is solved with varying values of R (and fixed N), an increased value of R naturally tends to give a better objective value (with $R = N$ the problem is unrestricted in terms of loading, while $R = 1$ gives a very restricted problem). However, it is *not* the case that, for a given problem instance, an optimal solution with a higher number of rows will always provide a lower bound on a solution with a lower number of rows (assuming that the length of the rows is always the minimum necessary to fit all items in the container, $L = \lceil \frac{N}{R} \rceil$).

As an example, consider an instance with 6 orders as shown in Figure B.1, where each graph constitutes a regular heptagon with 7 nodes, where 0 is the depot and the six orders are denoted by letters from A to F, such that the optimal TSP solution (the perimeter) in the pickup graph is (0, A, B, C, D, E, F, 0), while the optimal TSP solution in the delivery graph is (0, C, B, A, F, E, D, 0). The values c_{ij}^T are evaluated as Euclidean distances. In this case an optimal solution to this problem with 2 rows consists of the two optimal TSP tours and the row assignment {{A, B, C},{D, E, F}}. Figure B.1 shows the two graphs and the corresponding optimal loading plan with 2 rows (container seen from above).

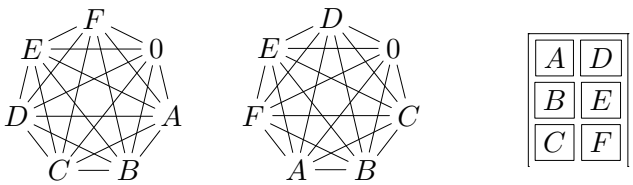


Figure B.1: Example where $z^*_{R=2} < z^*_{R=3}$.

When solving the same instance with 3 rows of length 2, no loading plan

exists which allows the use of the optimal TSP tour in both graphs, and hence its objective value will be strictly greater than that obtained with two rows.

If L is held constant, an optimal solution *will* always be a lower bound on the same instance solved with fewer rows, since one or more rows can then be left empty, meaning for example that a solution with 2 rows is still valid when using 3 rows.

B.3 Precedence models

In order to make the paper self-contained, we recall in this section the formulation that was first presented in [91]. We also present a variation of this formulation and, finally, a branch-and-cut approach to solve both models.

B.3.1 The precedence model

We call the formulation presented in [91] the *precedence model*. This formulation contains a polynomial number of constraints.

The variables are:

$$x_{ij}^T = \begin{cases} 1 & \text{if arc } (i,j) \text{ is used in graph } G^T, \\ 0 & \text{otherwise,} \end{cases}$$

$$y_{ij}^T = \begin{cases} 1 & \text{if item } i \text{ is handled before item } j, \\ 0 & \text{otherwise,} \end{cases}$$

$$z_i^r = \begin{cases} 1 & \text{if item } i \text{ is placed in row } r, \\ 0 & \text{otherwise.} \end{cases}$$

The objective function can then be expressed as:

$$\min \sum_{\substack{T \in \mathcal{T} \\ i,j \in V^T}} c_{ij}^T x_{ij}^T \tag{B.1}$$

and the constraints can be stated as follows:

$$\text{s.t.} \quad \sum_{i \in V^T} x_{ij}^T = 1 \qquad T \in \mathcal{T}, j \in V^T \tag{B.2}$$

$$\sum_{j \in V^T} x_{ij}^T = 1 \qquad T \in \mathcal{T}, i \in V^T \tag{B.3}$$

$$y_{ij}^T + y_{ji}^T = 1 \qquad T \in \mathcal{T}, i,j \in \mathcal{I} \tag{B.4}$$

$$y_{ik}^T + y_{kj}^T \leq y_{ij}^T + 1 \qquad T \in \mathcal{T}, i,j,k \in \mathcal{I} \tag{B.5}$$

$$x_{ij}^T \leq y_{ij}^T \qquad T \in \mathcal{T}, i,j \in \mathcal{I} \tag{B.6}$$

$$y^{\mathrm{P}}_{ij} + z^r_i + z^r_j \leq 3 - y^{\mathrm{D}}_{ij} \qquad r \in \mathcal{R}, i, j \in \mathcal{I} \tag{B.7}$$

$$\sum_{r \in \mathcal{R}} z^r_i = 1 \qquad i \in \mathcal{I} \tag{B.8}$$

$$\sum_{i \in \mathcal{I}} z^r_i \leq L \qquad r \in \mathcal{R} \tag{B.9}$$

$$x^T_{ij} \in \mathbb{B} \qquad T \in \mathcal{T}, i, j \in V^T, i \neq j \tag{B.10}$$

$$y^T_{ij} \in \mathbb{B} \qquad T \in \mathcal{T}, i, j \in \mathcal{I}, i \neq j \tag{B.11}$$

$$z^r_i \in \mathbb{B} \qquad r \in \mathcal{R}, i \in \mathcal{I}. \tag{B.12}$$

Constraints (B.2) and (B.3) are flow conservation constraints, stating that one unit of flow must enter and exit each node. Constraints (B.4) ensure that for each pair of items i, j the corresponding precedence variable must be set, i.e., either i is picked up before j or j is picked up before i. Constraints (B.5) express transitivity, that is, if i is before k and k is before j, then i must necessarily be before j. Constraints (B.6) make sure that if an arc (i, j) is used, then the precedence variable is set accordingly (i is visited before j, implying $y_{ij} = 1$). In the formulation of this set of constraints we make a slight abuse of notation, because the indices of the x variables should vary in the set of nodes while the indices of the y variables vary in the set of the items. Constraints (B.7) express the LIFO constraints, that only apply when two items are in the same row r; if i and j are placed in the same row ($z^r_i = z^r_j = 1$), and i is picked up before j ($y^{\mathrm{P}}_{ij} = 1$), then j must be delivered before i ($y^{\mathrm{D}}_{ij} = 0$ and thus $y^{\mathrm{D}}_{ji} = 1$). On a side note, this can also be expressed with the right hand side as $2 + y^{\mathrm{D}}_{ji}$, without affecting the behavior of the model. Finally, equations (B.8) ensure that all items are assigned to exactly one row, and constraints (B.9) enforce the row capacity L. Constraints (B.10)–(B.12) state that all variables are binary.

The presence of the y variables ensures that no subtours can exist, and thus no explicit subtour elimination constraints are needed in this formulation.

It can be observed that in the precedence formulation only the x variables appear in the objective function, and only the z variables carry information regarding the row assignment. The y variables do not carry any new information about the solution, and are given entirely by the values of the x variables. However, it is not possible to express any immediate connections between the x and z variables. Instead, each of the two sets of variables are tied to the y variables as illustrated in Figure B.2. This probably explains part of the difficulty of solving the model.

B.3.2 Variation: The row precedence model

A variation of the precedence model (B.1)–(B.12) has been developed in an attempt to find a more effective formulation that uses different sets of variables, and the resulting model is briefly described in the remainder of this section. We call the new model the *row precedence* model. Here, the y variables are replaced by a set of variables that additionally reflect the row assignment and is thereby sparser:

$$w_{ij}^r = \begin{cases} 1 & \text{if item } i \text{ is picked up before item } j \text{ and both are in row } r, \\ 0 & \text{otherwise.} \end{cases}$$

The model consists of the original objective function (B.1) and constraints (B.2), (B.3), (B.8) and (B.9), along with the following:

$$x(S) \leq |S| - 1 \qquad T \in \mathcal{T}, S \subseteq V^T, |S| \geq 2 \qquad \text{(B.13)}$$

$$w_{ij}^r + w_{ji}^r \geq z_i^r + z_j^r - 1 \qquad r \in \mathcal{R}, i, j \in \mathcal{I} \qquad \text{(B.14)}$$

$$2(w_{ij}^r + w_{ji}^r) \leq z_i^r + z_j^r \qquad r \in \mathcal{R}, i, j \in \mathcal{I} \qquad \text{(B.15)}$$

$$x(p_{ij}) + z_i^r + z_j^r \leq |p_{ij}| + 1 + w_{ij}^r \qquad r \in \mathcal{R}, i, j \in \mathcal{I}, p_{ij} \in \Pi \qquad \text{(B.16)}$$

$$x_{ij}^T \in \mathbb{B} \qquad T \in \mathcal{T}, i, j \in V^T, i \neq j \qquad \text{(B.17)}$$

$$w_{ij}^r \in \mathbb{B} \qquad r \in \mathcal{R}, i, j \in \mathcal{I}, i \neq j \qquad \text{(B.18)}$$

$$z_i^r \in \mathbb{B} \qquad r \in \mathcal{R}, i \in \mathcal{I}, \qquad \text{(B.19)}$$

where p_{ij} is a path from node i to node j either in the set V^{P} of the pickup graph or in the set V^{D} of the delivery graph, Π is the set of all paths p_{ij} and $|p_{ij}|$ is the number of arcs in the path p_{ij}. The notation $x(S)$ indicates the summation, over all the arcs that connect nodes of the set S, of the associated x variables. Similarly, $x(p_{ij})$ is the summation over all the arcs of the path p_{ij} of the associated x variables. Subtour elimination constraints (B.13) are necessary here since, contrary to the y variables, the w variables do not ensure the elimination of subtours. Constraints (B.14) ensure that if i and j are both in some row r, then a w variable must be set to one, and (B.15) similarly ensure that if i and j are in different rows, then no w can be set to one. Finally, (B.16) ensure that for all pairs i, j that are placed in the same row r, and for which some path from i to j exists (implying that i is before j), the corresponding w variable must be set to 1.

An additional variation of the precedence model can be derived, which uses all of the above-mentioned 4 variable sets, x, y, w, and z, in the hope that the additional ties between the variables would be beneficial. Thus, this model was constructed by adding all conceivable constraints for the variables. However, computational experiments showed that this variation was not effective.

B.3.3 A branch-and-cut approach

In order to speed up the solution of the precedence model, reduction of symmetry is performed by forcing the first item to be placed in the first loading row, the second in one of the two first loading rows, the third in one of the three first loading rows, and so on up to order $R - 1$.

To solve the precedence model and the row precedence model exactly, a branch-and-cut approach is used. In the case of the row precedence model presented in this section and of the majority of the models we present in this paper, the subtour elimination constraints (B.13) are relaxed in the branch-and-cut approach designed for their solution and a cut is added to the formulation when a violated constraint is identified. Whenever this happens we proceed in the following way. For each graph G^T and for each node i of the graph, $i = 1, \ldots, N$, a minimum cut from the depot to i is calculated on the values of the x variables. Here, a minimum cut is the optimal solution of the problem of partitioning the nodes into two non-empty connected components to minimise the total weight (in terms of x variables) of arcs whose end points are in different components, where one component contains the depot and the other one node i. In order to guarantee that there are no subtours, we need the minimum cut from the depot to any node i in each graph G^T to have a value not lower than 2. If a minimum cut has total value lower than 2, then a cut of the form (B.13) is added to the formulation. The minimum cut is obtained using the algorithm presented in [95]. At each iteration, when a constraint is identified as violated by the currently optimal solution in graph G^{P}, a cut is added to the formulation. Similarly, the constraints are checked in graph G^{D} and possibly another cut is added to the formulation. Then, a new iteration is run. At most one cut for each graph is added at each iteration. If no violated constraint is identified and the subtour elimination constraints were the only constraints that were relaxed in the original formulation, the optimal solution of the linear relaxation has been obtained. Otherwise, other constraints will be checked for violation.

The precedence model In the branch-and-cut approach to the precedence model, the large set of constraints (B.5) are removed from the formulation, and the constraints are added to the formulation when violated. At the first iteration the precedence model is solved without any of these constraints while in the subsequent iterations it is solved with those constraints that have been identified as violated at previous iterations. At each iteration all constraints (B.5) are checked and those that are violated by the currently optimal solution, that is the optimal solution of the linear relaxation of the current formulation, are added to the formulation. Several of these constraints may be added at each iteration.

Explicit subtour elimination constraints are not necessary in this model, as previously noted. However, they can be used to strengthen the for-

mulation. They can be checked for violation at each iteration and, if one is identified as violated, the corresponding cut can be added to the formulation.

In the section devoted to the computational tests, we will show the results of the tests that we carried out to evaluate the impact of the different combinations of possible cuts. When both (B.5) and (B.13) are used to dynamically add cuts in the branch-and-cut approach, both types are checked at each iteration.

The row precedence model The row precedence model cannot be formulated with a polynomial number of constraints, due to the presence of (B.13) and (B.16). In the branch-and-cut approach to this model, these constraints are relaxed initially and added as cuts when violated. Constraints (B.13) are subtour elimination constraints and are treated as described earlier in this section. At each iteration constraints (B.13) are checked first, and constraints (B.16) are checked only when all subtour elimination constraints are satisfied. Constraints (B.16) are checked for possible violation by checking each path of arcs with associated non-zero x variables. The paths are checked on both graphs, G^{P} and G^{D}. When a violated constraint is identified, it is added to the formulation. Several of these constraints may be added at each iteration. Although the number of these constraints grows exponentially with the size of the problem, for the instances that we tested in this paper we did not encounter any computational problems.

B.4 The flow model

An alternative formulation of the DTSPMS can be obtained by viewing the problem purely in terms of flows. In this section we present the *flow model* and the branch-and-cut approach adopted for its solution.

For this model the original graphs with node sets V^{P} and V^{D} are duplicated, adding a separate set of nodes for every loading row, creating a complete graph for the pickup and the delivery regions, respectively. A solution can be expressed as a number of flows in such a graph, with constraints expressing restrictions such as "if there is a flow on an arc a, there must be a flow on some path p". This idea is illustrated in Figures B.3–B.5 for an instance with $R = 2$ rows and $N = 6$ orders.

Figure B.3 illustrates the pickup route and row assignment of a feasible solution. The two rows are represented by the two copies of the graph (top and bottom), and the solid line represents the actual pickup route (0, 2, 3, 4, 5, 6, 1, 0). Each customer node is visited in exactly one of the subgraphs, indicating the loading row that this order is assigned to (row 1: orders 3, 4, 1; row 2: orders 2, 5, 6). The dotted lines in the figure indicate the order in which each row is loaded. A node must be

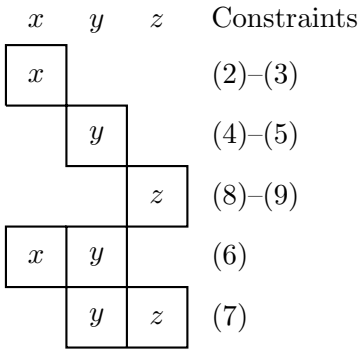


Figure B.2: Matrix structure of the precedence formulation (B.1)–(B.12).

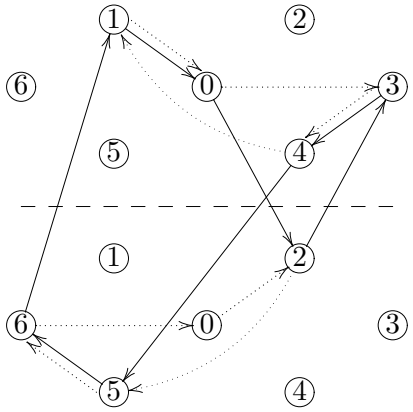


Figure B.3: An example of pickup graph used for the flow formulation.

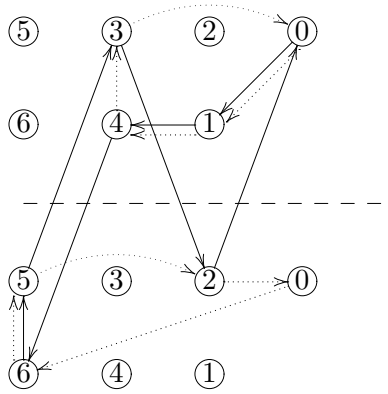


Figure B.4: A delivery graph for Figure B.3.

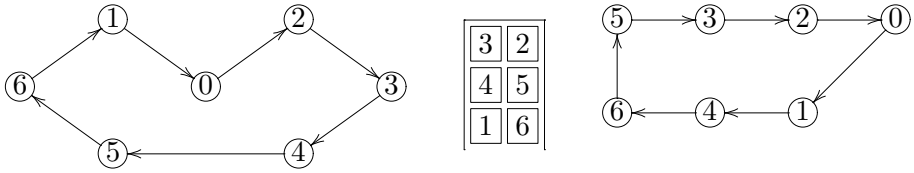


Figure B.5: The complete solution for the example of Figures B.3–B.4.

visited by either both "types" of flow (dashed and solid lines), or by no flow at all. Figure B.4 illustrates the corresponding delivery route, and Figure B.5 summarises the two routes and the loading plan.

Thus, each of the graphs $\tilde{G}^T$, over which the flow formulation is defined, is made of $2R$ subgraphs. In the pickup graph $\tilde{G}^{\mathrm{P}}$ there is a subgraph, that is a copy of G^{P}, for each row r. Similarly, for graph G^{D}. When defining the problem variables, a node is a copy of a node, associated with a row, of graph G^{P} or G^{D}. A node in the graph $\tilde{G}^{\mathrm{P}}$ is identified by a pair (i, r), where i is the order and r is the row. Similarly, for graph $\tilde{G}^{\mathrm{D}}$. The set S of nodes in (B.23) consists of all such pairs.

The *flow model* uses two sets of binary variables, both of which are flow variables:

- $x_{ij}^{Tpq} = 1$, iff the arc between nodes (i, p) and (j, q) is travelled in $\tilde{G}^T$, i.e., item i is picked up immediately before j, i is in row p, and j is in row q;
- $v_{ij}^{r} = 1$, iff the arc between nodes (i, r) and (j, r) is travelled in the pickup subgraph of row r, i.e., i and j are both in row r, and no other item of row r is picked up between i and j.

In Figures B.3 and B.4 the solid lines correspond to the x variables, while the dotted lines correspond to the v variables.

The x variables describe a flow through the entire graph representing the actual travelled route, and are thus again the only variables appearing in the objective function. The use of the x variables in this model is closely related to that in the precedence model, in that $x_{ij}^{Tpq} = 1$ in the flow model if and only if $x_{ij}^{T} = z_i^p = z_j^q = 1$ in the precedence model. Thus, the complete solution to an instance can here be expressed solely in terms of the x variables, since these now also contain the row assignments.

The v variables describe the flow in each of the subgraphs, representing the order in which the items in a given row are loaded, and are required to express the LIFO connection for each row between the pickup and delivery routes.

Each node in the graph must be passed by the same amount of x and v flow, and the difficulty in this model lies in ensuring that the order of any pair of customers is the same on the x and v flows.

In the formulation that follows, we refer to the copies of the original graphs G^{P} and G^{D}, one for each row. We will use the notation $S \subseteq (V^T \times \mathcal{R})$ to indicate that S is a subset of nodes of the graph $\tilde{G}^T$, where each node is a pair (i, r), with $i \in V^T$ and $r \in \mathcal{R}$.

The objective function can now be expressed as:

$$\min \sum_{\substack{T \in \mathcal{T} \\ i,j \in V^T}} \left(c_{ij}^{T} \sum_{p,q \in \mathcal{R}} x_{ij}^{Tpq} \right) \tag{B.20}$$

and the constraints can be stated as follows:

$$\sum_{\substack{p,q\in\mathcal{R}\\ j\in V^T}} x_{ij}^{Tpq} = \sum_{\substack{p,q\in\mathcal{R}\\ j\in V^T}} x_{ji}^{Tqp} = 1 \qquad T\in\mathcal{T}, i\in V^T \tag{B.21}$$

$$\sum_{\substack{q\in\mathcal{R}\\ j\in V^T}} x_{ij}^{Trq} = \sum_{\substack{q\in\mathcal{R}\\ j\in V^T}} x_{ji}^{Tqr} \qquad T\in\mathcal{T}, r\in\mathcal{R}, i\in V^T \tag{B.22}$$

$$x(S) \leq |S| - 1 \qquad S\subseteq (V^T\times\mathcal{R}), |S|\geq 2, T\in\mathcal{T} \tag{B.23}$$

$$\sum_{j\in\mathcal{R}} v_{ij}^r = \sum_{j\in\mathcal{R}} v_{ji}^r \qquad r\in\mathcal{R}, i\in\mathcal{R} \tag{B.24}$$

$$\sum_{i,j\in\mathcal{R}} v_{ij}^r \leq L+1 \qquad r\in\mathcal{R} \tag{B.25}$$

$$\sum_{\substack{q\in\mathcal{R}\\ j\in V^T}} x_{ji}^{\mathrm{P}qr} = \sum_{\substack{q\in\mathcal{R}\\ j\in V^T}} x_{ij}^{\mathrm{D}rq} = \sum_{j\in\mathcal{R}} v_{ij}^r \qquad r\in\mathcal{R}, i\in V^T, T\in\mathcal{T} \tag{B.26}$$

$$\sum_{j\in\mathcal{R}} v_{0j}^r = 1 \qquad r\in\mathcal{R} \tag{B.27}$$

$$\sum_{r\in\mathcal{R},j\in\mathcal{R}} v_{ij}^r = 1 \qquad i\in\mathcal{R} \tag{B.28}$$

$$\sum_{r\in\mathcal{R},j\in\mathcal{R}} v_{ji}^r = 1 \qquad i\in\mathcal{R} \tag{B.29}$$

$$v_{ij}^r + |p_{ij}^r| - 1 \geq x(p_{ij}^r) \qquad r\in\mathcal{R}, p_{ij}^r\in\Pi^r \tag{B.30}$$

$$x_{ij}^{Tpq} \in \mathbb{B} \qquad T\in\mathcal{T}, p,q\in\mathcal{R}, i,j\in V^T, i\neq j \tag{B.31}$$

$$v_{ij}^r \in \mathbb{B} \qquad r\in\mathcal{R}, i,j\in\mathcal{R}, i\neq j. \tag{B.32}$$

The set Π^r contains all x-paths p_{ij}^r from i to j, that start and end in the subgraph associated with row r and do not contain any intermediate nodes in this subgraph, implying that $v_{ij}^r = 1$.

Constraints (B.21) ensure that each customer has one unit of flow in and out. Constraints (B.22) ensure balance of x flow for each node in the graph, while (B.24) ensure balance of v flow. Constraints (B.23) are subtour elimination constraints on the x variables. Row capacities are enforced by (B.25) (which hold with equality when $N = R \times L$); in each subgraph the tour must consist of (at most) L items plus the depot. Constraints (B.26) ensure that the x and v flows exiting a node are equal, i.e., if a node is passed by one type of flow it is passed by both types. Through (B.27) it is guaranteed that the depot is visited in each subgraph, which is necessary to ensure that there is a well-defined starting point, when determining whether one node is visited *before* some other node. Constraints (B.28) and (B.29) ensure that each customer is visited exactly once. Constraints (B.30) state that if an x-path $p_{ij}^r \in \Pi^r$

exists between customers i and j, then the corresponding v variable must be set to one, and (B.31)–(B.32) are the domains of the variables.

Constraints (B.30) would be expected to favor instances with fewer rows, since the average length of any x-path in Π^r before returning to its starting row will be shorter in these instances. In problems with many loading rows a path is more likely to visit many customers before returning to its starting row. Additionally the total number of nodes in the graph will depend on the number of rows, which should further favor instances with fewer rows.

B.4.1 A branch-and-cut approach

When solving the flow model with a branch-and-cut approach, the subtour elimination constraints (B.23) and the *connection cuts* (B.30) are relaxed initially and these cuts are added to the formulation when violated constraints are identified.

The constraints (B.30) are separated as follows (the description is given for $\tilde{G}^{\mathrm{P}}$ only, and is similar for $\tilde{G}^{\mathrm{D}}$):

- select a pair of nodes i and j such that v^r_{ij} has value 1 in the currently optimal solution;
- follow a path of x variables with value 1 from node (r, i) of graph $\tilde{G}^{\mathrm{P}}$, until a node $(r, \cdot)$ is reached;
- if the reached node is not (r, j), it means a violation has been identified and the corresponding violated constraint (B.30) is added as a cut to the formulation.

Even for non-integral solutions it would be possible to identify some violated cuts of type (B.30), with values $0 < v^r_{ij} \leq 1$. However, in this case the separation would be quite cumbersome, and we have decided to only search for violated connection cuts when an integer solution is at hand.

B.5 The Travelling Salesman Problem with Infeasible Paths model

In this section we present a model that maintains some of the modelling ideas of the precedence model, but faces the modelling of the loading aspect in a substantially different way, avoiding the explicit use of the y variables and instead focusing on the construction of two optimal TSP tours that allow a feasible solution in terms of loading and unloading of the vehicle. This idea is inspired by the use of infeasible paths in [65], and we call this model the *Travelling Salesman Problem with Infeasible Paths* (TSPIP) model. The model is based on the idea of decomposing the problem into a master problem which focuses on the routing aspects, and a subproblem focusing on the loading aspects. Thus the task of

the master problem becomes to build two TSP tours, one for the pickup operations and one for the delivery operations, using only the x variables of the precedence model. The subproblem should then identify cuts that eliminate any set of tours that do not allow the construction of a feasible loading plan. More precisely, when the two TSP tours are found by the master problem, the corresponding values of the precedence variables of the precedence model can be calculated and used as parameters in the subproblem, to determine if a feasible loading plan exists. If this is not the case, the subproblem will produce a cut in the form of an infeasible path that is violated by the current solution, which can be added to the master problem before this is solved again. We call the master problem the *TSPIP-master* and use the term *TSPIP-subproblem* to refer to the loading feasibility subproblem, that checks whether a feasible loading plan exists and, in case it does not, produces a cut.

We first present the TSPIP model and then the solution approach.

B.5.1 The TSPIP model

The TSPIP model employs the arc variables x_{ij}^T used in the precedence model and is formulated as:

$$\min \sum_{\substack{T \in \mathcal{T} \\ i,j \in V^T}} c_{ij}^T x_{ij}^T \tag{B.33}$$

subject to the constraints:

$$\sum_{i \in V^T} x_{ij}^T = 1 \qquad T \in \mathcal{T}, j \in V^T \tag{B.34}$$

$$\sum_{j \in V^T} x_{ij}^T = 1 \qquad T \in \mathcal{T}, i \in V^T \tag{B.35}$$

$$x(S) \leq |S| - 1 \qquad T \in \mathcal{T}, S \subseteq V^T, |S| \geq 2 \tag{B.36}$$

$$x(p) \leq |p| - 1 \qquad p \in P \tag{B.37}$$

$$x_{ij}^T \in \mathbb{B} \qquad T \in \mathcal{T}, i, j \in V^T, i \neq j. \tag{B.38}$$

The flow conservation constraints (B.34) and (B.35) are identical to (B.2) and (B.3), and constraints (B.36) are regular subtour elimination constraints, which are now necessary since the y variables are no longer present.

The special feature of the TSPIP model is the presence of constraints (B.37) that eliminate all pairs of pickup and delivery tours that do not allow the construction of a feasible loading plan. The set P is the set of all *loading infeasible* paths. In this section a *path* is a sequence of customers that include both pickup and delivery customers. A path contains the last n_{P} customers of the pickup tour and the first n_{D} customers of the delivery tour and a *loading infeasible* path is a path for which no feasible

loading plan exists. Take, for example, an instance with $R = 1$. Then, a path where the last customers of the pickup tour are customers 1 and 2 and the first customers of the delivery tour are customers 1 and 2 is loading infeasible because, in the delivery tour, customer 1 has to be served after customer 2. The expression $x(p)$ indicates the sum, over all the arcs of p, of the x variables associated with the arcs. The quantity $|p|$ denotes the number of arcs in path p.

Obviously, the enumeration of all loading infeasible paths of P is in general very cumbersome, and the formulation lends itself to a decomposition approach.

B.5.2 The solution approach

The decomposition approach used for the solution of the TSPIP model is based on the solution of the TSPIP-master, which finds a minimum length pair of tours (a pickup tour and a delivery tour), and the TSPIP-subproblem, which checks whether a feasible loading plan exists for that pair of tours and, in case it does not, identifies a loading infeasible path p and a corresponding cut $x(p) \leq |p| - 1$. Then, the cut is added to the TSPIP-master that will find a new pair of tours, of the same or longer length. The procedure is repeated until the TSPIP-subproblem verifies that a feasible loading plan exists for a pair of tours. In this case, the given pair of tours is optimal.

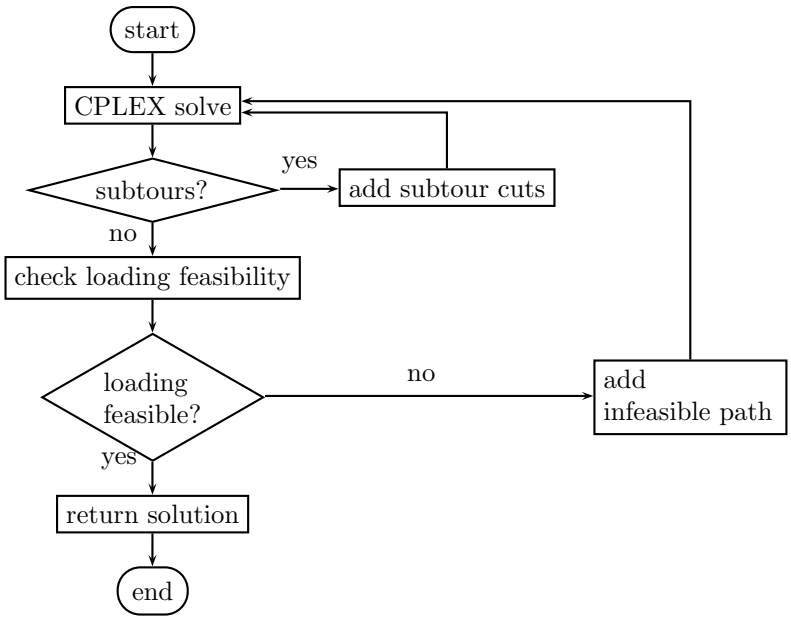


Figure B.6: Flow chart of the solution approach to the TSPIP.

At each iteration of this solution approach, the TSPIP-master formula-

tion is solved. The formulation coincides with the TSPIP model where, in the constraints (B.37), the set P is replaced by the subset P' that contains the loading infeasible paths identified by the TSPIP-subproblem in the previous iterations. The optimal solution of the TSPIP-master is obtained through a branch-and-cut approach where the subtour elimination constraints are initially relaxed and a cut is added for each graph at each iteration when violation is identified (see Section B.3.3). In the beginning the set P' is empty and the TSPIP-master identifies the unconstrained pair of tours of minimum cost. At each iteration of the solution approach, new paths are identified by the TSPIP-subproblem and new cuts are added to P' until the TSPIP-subproblem finds a loading feasible plan. The solution of the TSPIP-master is obtained through a branch-and-cut approach that completely relaxes the subtour elimination constraints at the beginning and then adds violated constraints.

Whenever the TSPIP-master has produced a subtour-free optimal pair of tours, the TSPIP-subproblem is solved. It will either return one or more infeasible paths to be added to the TSPIP-master, or determine that none exist and that the current solution is thereby feasible and optimal. The structure of the solution approach to the TSPIP is depicted in Figure B.6.

The TSPIP-subproblem

We now present the algorithm that is used to solve the TSPIP-subproblem. The variables that are used to model the TSPIP-subproblem are identical to the z variables of the precedence model:

$$z_i^r = \begin{cases} 1 & \text{if item } i \text{ is in row } r, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, the TSPIP-subproblem uses a set of parameters $\bar{y}$, which correspond to the values of the y variables of the precedence model, and can be calculated after the two TSP tours have been obtained through the solution of the TSPIP-master:

$$\bar{y}_{ij}^T = \begin{cases} 1 & \text{if } i \text{ is before } j \text{ in tour } T \text{ of the current solution,} \\ 0 & \text{otherwise.} \end{cases}$$

Since there are no costs connected to the loading plan, the subproblem is a feasibility problem and does not have an objective function.

The constraints of the TSPIP-subproblem are identical to constraints (B.7)–(B.9) of the precedence model, with variables y_{ij}^T replaced by pa-

rameters $\bar{y}_{ij}^{T}$:

$$z_i^r + z_j^r \leq 3 - \bar{y}_{ij}^{\mathrm{P}} - \bar{y}_{ij}^{\mathrm{D}} \qquad r \in \mathcal{R}, i, j \in \mathcal{I} \tag{B.39}$$

$$\sum_{r \in \mathcal{R}} z_i^r = 1 \qquad i \in \mathcal{I} \tag{B.40}$$

$$\sum_{i \in \mathcal{I}} z_i^r \leq L \qquad r \in \mathcal{R} \tag{B.41}$$

$$z_i^r \in \mathbb{B} \qquad r \in \mathcal{R}, i \in \mathcal{I}. \tag{B.42}$$

Note that only those constraints (B.39) for which $\bar{y}_{ij}^{\mathrm{P}} = 1$ and $\bar{y}_{ij}^{\mathrm{D}} = 1$ need to be included.

The TSPIP-subproblem can be seen as an l-bounded k-coloring problem. Constraints (B.39) are active only when $\bar{y}_{ij}^{\mathrm{P}}$ and $\bar{y}_{ij}^{\mathrm{D}}$ are both equal to one. Then, considering k equal to the number of rows and $l = L$, we can construct a graph $G' = (V', E')$ where $V' = V$ and, for each pair of nodes i and j, there exists an edge joining them if and only if $\bar{y}_{ij}^{\mathrm{P}} = \bar{y}_{ij}^{\mathrm{D}} = 1$. Thus, finding an l-bounded k-coloring on G' corresponds to finding a feasible solution to the TSPIP-subproblem. However, the parameters $\bar{y}_{ij}^{T}$ have a specific structure since they represent precedence relations in a tour. Thus, even if it possible to prove that graph G' does not have a specific structure (for example, it can be connected or disconnected, acyclic or contain cycles), a polynomial transformation from the l-bounded k-coloring problem to the TSPIP-subproblem is not straightforward. In any case, since the l-bounded k-coloring is proven to be NP-complete (see [38]), our conjecture is that the TSPIP-subproblem is NP-complete.

As any infeasibility cut for a path is dominated by the infeasibility cut for any sub-path, we aim at finding the shortest possible infeasible paths. For this reason, the TSPIP-subproblem is solved by repeatedly solving model (B.39)–(B.42) with an increasing number of orders, thus examining potential infeasible paths of increasing length.

Note that, if in a path $n_{\mathrm{P}} + n_{\mathrm{D}} \leq R$, the path is certainly feasible. More specifically, for a path to possibly be loading infeasible it must contain at least $R + 1$ distinct orders, with at least one pickup and one delivery customer.

The search for infeasible paths begins by constructing a path consisting of the $n_{\mathrm{P}} = \lceil R/2 \rceil$ last pickup customers and the $n_{\mathrm{D}} = \lceil R/2 \rceil$ first delivery customers. This is the shortest path that can possibly be loading infeasible, and contains a total of at most $R + 1$ distinct customers. If it contains fewer than $R + 1$ distinct customers it is extended before the first check is performed.

Furthermore, for the TSPIP-subproblem (B.39)–(B.42) to be well defined all included orders must contain both a pickup and a delivery customer. Indeed, it is necessary to have both pickup and delivery customers for each order in order to satisfy constraints (B.39). This is generally not

the case when taking the last part of the pickup route and the first part of the delivery route. Typically, one or more pickup customers will be present, whose corresponding delivery customer is not in the path, and vice versa. In this case, we need to extend the path to include the missing customers. Let us consider, for example, the pickup path where some customers are missing, i.e., they are not present in the pickup path but the corresponding delivery customers are present in the delivery path. In order to solve the TSPIP-subproblem (B.39)–(B.42), we want to add the missing pickup customers somewhere in the pickup path. The missing customers have to be placed at the start of the pickup path, as otherwise we would modify the TSP solution which we are checking for a possible infeasibility in the loading plan. Moreover, we have to consider the order of insertion of the missing customers at the start of the path. In order to avoid the introduction of any infeasibility caused by the addition of the missing customers, they have to be added in the order *opposite* to the order of appearance of the corresponding orders in the delivery path. In this way, the TSPIP-subproblem on the new path (the extended sequence of pickup customers combined with the original sequence of delivery customers) is infeasible if and only if the sequence of $n_{\mathrm{P}} + n_{\mathrm{D}}$ customers of the original path is infeasible, and hence feasibility is not affected by the addition of the missing customers. A similar procedure is used when extending the delivery part of the path with customers that only appear for pickup in the subproblem. Following this approach, for example, the path $\{1, 2 \mid 3, 1, 4\}$ (where $\mid$ shows the split between pickup and delivery customers) would become $\{4, 3, 1, 2 \mid 3, 1, 4, 2\}$.

Note that, if the distances are symmetric, whenever a loading infeasible path is identified, another relevant loading infeasible path can be found by reversing the order of both the pickup and the delivery parts of the path and solving the subproblem again. Thus, for every solution to the TSPIP-master, the subproblem can be solved twice to produce two different cuts, before returning to the TSPIP-master. This idea exploits the symmetry of the distance matrix, where the pair of reversed routes will produce another optimal solution to the master problem.

Other ways to simultaneously identify several cuts can be considered. Figure B.7 shows how several loading infeasible paths can exist for a given solution to the TSPIP-master. The numbers on the top indicate a path, and the lines indicate paths that are loading infeasible for a problem with $R = 2$.

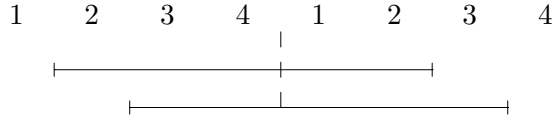


Figure B.7: Different infeasible paths for the same TSPIP-master solution ($R = 2$).

In order to systematically identify such cuts one would need to extend

the path in one route only, while the part of the path that lies in the other route is kept constant, until a violation is found (or it can be determined that a dominating cut has already been found). This could then be repeated for all possible constant partial paths up to length N. However, such an approach has not been pursued in this paper.

As our conjecture is that the TSPIP-subproblem is NP-hard, we tested the time needed to solve it using a standard solver. In fact, the time is systematically extremely short. Only in very few cases is it of the order of one second, whereas in most cases it is insignificant. The number of times that the TSPIP-subproblem is invoked depends on the size of the instance. On the tested instances, the number ranges from a few hundred on the small instances, to more than 100,000 times on the largest tested instances.

B.6 Computational results

Each of the models presented in Sections B.3, B.4 and B.5 has been implemented using Java 1.5 and ILOG Concert Technology with CPLEX version 9.1. We used the standard settings of CPLEX, and used nested callbacks for adding cuts. All test runs have been performed on an Intel Pentium 4, 2.8 GHz with 2 GB RAM. Unless otherwise mentioned, all tests have been performed with a maximum allowed running time of one hour.

All tests have been performed on instances from the set of 20 randomly generated instances from [91], which can be found online[1]. We will specify the number and size of the tested instances for each test. The data for each instance of a given size has been obtained by taking the necessary number of customers from the beginning of each file. The notation for the size of the tested instances throughout this section is $R \times L$, for R rows of length L.

For all test runs on instances with $N = R \times L$, an initial solution to the problem has been supplied. This initial solution has been obtained using the simulated annealing heuristic described in [91] with 3 minutes of allowed running time. In cases where $N < R \times L$ (Sections B.6.5 and B.6.6) this heuristic is unable to produce solutions, and therefore no initial solution was provided.

Initially, some experiments were performed to examine the behavior of CPLEX when more time is allowed. The results can be seen in Table B.2, which reports the progress over time, by listing the current MIP gap every hour over a twelve hour period for one instance of each of three different problem sizes. The model tested is the precedence model (B.1)–(B.12). All three columns of the table show the slow progress of the lower bound.

[1] `http://www.transport.dtu.dk/datasets/DTSPMS`

Hrs	2×7	3×7	4×7
1	7.60%	9.33%	11.89%
2	7.31%	9.28%	11.89%
3	7.12%	9.26%	11.89%
4	6.99%	9.17%	11.89%
5	6.90%	9.13%	11.71%
6	6.82%	9.10%	11.71%
7	6.76%	9.08%	11.71%
8	6.71%	9.06%	11.71%
9	6.65%	9.03%	10.62%
10	6.61%	9.00%	10.62%
11	6.57%	8.97%	10.62%
12	6.53%	8.94%	10.62%

Table B.2: MIP gap obtained by CPLEX, progress over time.

B.6.1 Impact of Cuts

In this section the impact of relaxing some of the constraints and adding them when violated is examined, as well as the impact of strengthening cuts.

Table B.3 shows the results for the precedence model. The tests have been carried out for four different instance sizes, and each number shows the average MIP gap over five instances after one hour of running time. The first line of the table shows the results for the complete model, as presented in (B.1)–(B.12) and directly solved by CPLEX. The following lines show the impact of using different sets of constraints to add as cuts when violated.

	2×7	3×7	4×7	4×5
Precedence model (B.1)–(B.12)	7.98%	12.12%	17.17%	5.74%
- with (B.5) only added as cuts	7.88%	11.64%	16.95%	4.63%
- with subtour elimination cuts only	7.82%	9.89%	12.08%	4.09%
- with both added as cuts	7.71%	9.92%	12.09%	4.02%

Table B.3: Impact of using cuts and additional inequalities in the precedence model.

First, constraints (B.5) are removed from the model, and at each node are checked for violation and added accordingly. In the following line, constraints (B.5) are kept in the formulation from the beginning while subtour elimination constraints are used as valid inequalities, checked for violation and are added as cuts when violated. In the last line both sets of constraints are used to generate cuts. The results indicate that particularly the subtour elimination constraints are valuable, whereas the effect of adding (B.5) as cuts is reduced when combined with the subtour elimination constraints. We will use both types of cuts for the rest of this paper.

When the flow model (B.20)–(B.32) is solved, constraints (B.23) and (B.30) are removed and inserted when violated due to their number. However, constraints of type (B.30) can be added directly if $|p^r_{ij}|$ is small. The results of tests that include (B.30) for small values of $|p^r_{ij}|$ do not show any practical advantage of this.

B.6.2 Comparison of models

Table B.4 reports the results of running each of the three models on five instances of different sizes and configurations. The first column gives the problem instance: instance name and loading configuration ($R \times L$). Here n is the total number of graph nodes, Opt. is the value of the optimal solution, when known, and UB is the value of the heuristic initial solution supplied to the algorithm. Whenever a value of Opt. is provided for an instance which no tested method could solve to optimality within the fixed time limit, the solution has been found by allowing CPLEX to run for a longer time. For each of the tested models there are two columns, presenting the MIP gap after one hour of running time for each instance, and the solution time in seconds (3600 seconds being the maximum allowed running time). As can be seen from the table, the TSPIP model performs significantly better than the other models, solving more instances, and leaving a smaller MIP gap on the remaining instances. Additionally, the precedence model performs better than the flow model on the small and medium sized instances, while the performance of the two is comparable for the larger instances. On the largest set of instances (4×7) all three models produce very similar results.

In a few cases it can be noticed that the resulting MIP gap is reduced when R is increased by 1 (with L fixed), even though this means that the number of orders is increased by 33–50%. A likely cause is that the nature of the solution changes considerably when the number of rows changes, and that the difficulty of solving different instances of a given size can also vary significantly. A similar effect is never seen when the row length is increased.

B.6.3 Precedence model variations

We now compare the precedence models of Section B.3 with its variation.

Table B.5 shows the average MIP gap over 5 instances of each size for the precedence model and the row precedence model.

These results indicate that the addition of the row index to the precedence variable did not improve the performance of the model.

Instance	n	Opt.	UB	Precedence Model		Flow Model		TSPIP	
				gap	time/s.	gap	time/s.	gap	time/s.
R05_2×4	18	501	501	0%	< 1	0%	4	0%	< 1
R06_2×4	18	694	694	0%	20	0%	303	0%	31
R07_2×4	18	487	487	0%	5	0%	302	0%	27
R08_2×4	18	642	642	0%	53	0%	388	0%	38
R09_2×4	18	558	558	0%	11	0%	81	0%	17
R05_2×5	22	546	546	0%	69	1.13%	3600	0%	196
R06_2×5	22	774	774	0%	215	4.04%	3600	0%	678
R07_2×5	22	547	547	0%	58	0%	2246	0%	115
R08_2×5	22	670	670	0%	290	3.06%	3600	0%	392
R09_2×5	22	610	610	0%	16	0%	381	0%	44
R05_2×6	26	-	631	7.13%	3600	11.09%	3600	7.92%	3600
R06_2×6	26	793	793	4.30%	3600	6.73%	3600	2.77%	3600
R07_2×6	26	593	593	5.03%	3600	8.87%	3600	4.05%	3600
R08_2×6	26	749	749	7.34%	3600	9.63%	3600	4.65%	3600
R09_2×6	26	692	692	2.38%	3600	3.10%	3600	0%	1680
R05_2×7	30	-	775	2.86%	3600	8.13%	3600	7.08%	3600
R06_2×7	30	-	824	10.98%	3600	7.65%	3600	5.22%	3600
R07_2×7	30	-	697	11.24%	3600	11.19%	3600	6.95%	3600
R08_2×7	30	-	824	7.07%	3600	11.65%	3600	8.01%	3600
R09_2×7	30	739	739	9.59%	3600	3.36%	3600	1.53%	3600
R05_3×4	26	567	567	0%	38	1.06%	3600	0%	7
R06_3×4	26	747	747	0%	79	1.47%	3600	0%	15
R07_3×4	26	557	557	0%	804	3.77%	3600	0%	90
R08_3×4	26	690	690	0%	112	2.03%	3600	0%	7
R09_3×4	26	669	672	0%	31	0.74%	3600	0%	6
R05_3×5	32	737	737	2.58%	3600	2.85%	3600	0%	2442
R06_3×5	32	836	836	2.51%	3600	2.99%	3600	0%	1815
R07_3×5	32	690	690	4.96%	3600	5.65%	3600	2.39%	3600
R08_3×5	32	826	826	3.06%	3600	4.48%	3600	0%	2028
R09_3×5	32	768	768	1.26%	3600	2.28%	3600	0%	666
R05_3×6	38	804	833	9.50%	3600	7.86%	3600	6.89%	3600
R06_3×6	38	-	871	6.39%	3600	4.25%	3600	3.56%	3600
R07_3×6	38	-	758	8.85%	3600	8.97%	3600	8.08%	3600
R08_3×6	38	-	864	5.21%	3600	5.44%	3600	4.61%	3600
R09_3×6	38	774	796	5.90%	3600	4.40%	3600	0%	1995
R05_3×7	44	-	900	11.12%	3600	9.61%	3600	7.64%	3600
R06_3×7	44	-	949	11.80%	3600	10.54%	3600	10.41%	3600
R07_3×7	44	-	841	9.74%	3600	9.75%	3600	9.45%	3600
R08_3×7	44	-	916	10.60%	3600	9.72%	3600	9.53%	3600
R09_3×7	44	-	891	15.30%	3600	10.33%	3600	10.21%	3600
R05_4×4	34	744	750	4.37%	3600	3.07%	3600	0%	1271
R06_4×4	34	821	841	0.73%	3600	3.45%	3600	0%	281
R07_4×4	34	673	673	0%	341	0.45%	3600	0%	3
R08_4×4	34	815	819	1.95%	3600	2.08%	3600	0%	123
R09_4×4	34	755	774	0%	648	2.45%	3600	0%	1
R05_4×5	42	825	856	7.61%	3600	6.48%	3600	5.57%	3600
R06_4×5	42	859	894	6.00%	3600	4.92%	3600	0%	2347
R07_4×5	42	763	795	6.42%	3600	6.42%	3600	0%	3542
R08_4×5	42	841	853	3.54%	3600	3.28%	3600	0%	1614
R09_4×5	42	796	818	1.07%	3600	2.69%	3600	0%	4
R05_4×6	50	-	933	12.43%	3600	9.65%	3600	9.62%	3600
R06_4×6	50	-	975	12.34%	3600	10.87%	3600	10.77%	3600
R07_4×6	50	-	916	14.42%	3600	12.01%	3600	11.90%	3600
R08_4×6	50	-	924	7.40%	3600	5.41%	3600	5.25%	3600
R09_4×6	50	-	882	14.68%	3600	6.18%	3600	6.04%	3600
R05_4×7	58	-	984	18.08%	3600	11.89%	3600	11.88%	3600
R06_4×7	58	-	1034	14.07%	3600	11.17%	3600	10.93%	3600
R07_4×7	58	-	1002	17.56%	3600	12.57%	3600	12.44%	3600
R08_4×7	58	-	1088	19.58%	3600	15.49%	3600	15.26%	3600
R09_4×7	58	-	975	14.87%	3600	9.64%	3600	9.63%	3600

Table B.4: Comparison of models.

	3×5	4×7
Original precedence model	2.26%	12.06%
Row precedence variation	3.57%	12.14%

Table B.5: Comparison of precedence model variations.

B.6.4 Number of rows vs. length of rows

Somewhat surprisingly, the length of the rows in the problem has proven to be much more significant than the number of rows in terms of difficulty of solving the problem.

This trend can already be observed in Table B.4, and a further illustration is given in Table B.6, which shows results obtained by solving 10 instances with 18 orders in different loading configurations, using the TSPIP approach. For each instance size the table reports the number of instances (out of 10) that were solved to optimality within one hour, the average MIP gap after one hour of computation and the average solution time used.

	2×9	3×6	6×3	9×2	18×1
Optimal	0	1	10	10	10
Gap (avg.)	11.96%	5.24%	0%	0%	0%
Time (avg.)/s.	3600	3283	5.5	4.6	2.1

Table B.6: Instances with 36 customer nodes in different configurations.

The table clearly shows that instances with many short rows are significantly easier to solve than instances with only a few long rows. The same trend was observed during other tests, and when attempting to determine optimal solutions (allowing longer running times) to the test problems.

B.6.5 Extra loading capacity

The instances considered so far have dealt with loading plans where the number of available loading positions equals the number of items to be loaded, i.e., $N = R \times L$. However, one could also examine instances where the number of loading positions is allowed to be greater than the number of items. The purpose of this is twofold: 1) to determine if this change has any impact on the difficulty of solving the problem, and 2) from a business point of view, to determine if any cost reduction can be achieved by reducing the loading degree, i.e., by running the vehicles with some spare capacity to increase loading flexibility.

In this section all instances have been solved without providing an initial solution, because the heuristic used to produce initial solutions could not solve instances with $N < R \times L$. This option was preferred over adopting a solution obtained for $N' = R \times L$ where $N' \geq N$, since the effect of

such an adopted solution might disturb the results, as the quality of the initial solution will be relatively better when the difference $R \times L - N$ is small. Naturally, for actual solutions it will always be beneficial to provide an initial solution.

Since it is clearly not possible to gradually increase the loading capacity in steps of size one, the tests have been conducted by stepwise increases of the row length, which gives the smallest meaningful increase of the loading capacity.

These tests have been performed on 10 instances with 15 orders loaded as 3×5, 3×6, 3×7, and 3×8, and the results can be seen in Table B.7. The table provides the row length L, the number of instances solved to optimality, the average time to solve those instances and the average gap after one hour. The results indicate that instances with some extra space could be slightly easier to solve. However, adding large amounts of extra space does not seem to provide any further advantage.

L	Opt.	Time (avg.)/s.	Gap
5	6	2017	0.63%
6	7	1964	0.80%
7	6	1532	0.58%
8	6	1520	0.57%

Table B.7: Tests with extra space.

For the instances with $L = 6$ more instances can be solved to optimality in a shorter time than for $L = 5$. However, the average solution quality does not improve. This is caused by one instance which becomes much more difficult.

Concerning the possibility of cost reduction there does not seem to be a significant gain achievable by increasing the loading capacity. When the row length increases from 5 to 6, 5 instances show objective value decreases by 1.2%, 0.4%, 0.2%, 1.9% and 0.4%, while the remaining are unchanged. With the additional increase of the row length to 7 only one decrease of 1.0% occurs, and when increasing from 7 to 8 no gain is achieved. Thus, it seems unlikely that any significant gain would be obtainable in practice by allowing extra capacity.

B.6.6 Instances from the literature

In addition to the randomly generated instances from [91], the TSPIP approach has been applied to some pickup and delivery instances derived from TSPLIB, namely the instances that were adopted by [11] to test the TSPPDL[2]. These instances provide node coordinates, as well as a division of the set of customers into pickup and delivery customers, and a 1-to-1 matching of the pickup and delivery customers. The instances

[2] `http://neumann.hec.ca/chairelogistique/data/TSPPDL-BB/`

contain 19, 23, 27, 31, 35, 39, 43, 47, and 51 nodes, meaning 9, 11, 13, 15, 17, 19, 21, 23, and 25 orders, and have been solved here with 3 loading rows. Since the DTSPMS requires a depot in each graph, the same location has been used in both graphs. This means that a given instance always contains one more node when considering the DTSPMS than when considering the TSPPDL, since the depot is used twice. The matching of pickup and delivery customers that is provided with the instances has also been used here.

Table B.8 shows for each instance the name, the total number of nodes n, the number of orders N, the loading configuration as $R \times L$, the final upper and lower bounds, the gap between the two, and the solution time. No solution time is reported for instance nrw1379 with 25 orders, since this instance finished prematurely with an out-of-memory error. The TSPIP approach has been provided with a heuristic initial solution whenever possible ($N = R \times L$); these instances have been marked by a $^+$ in the table. All other instances have been solved without providing the solver with an initial solution, and in these cases we see that the algorithm often fails to find a feasible solution.

It can be noted that the results given in Table B.8 are generally consistent with those of Table B.4, regarding the size of the instances that can be solved, with a slightly larger variation, as only 8 out of 9 instances are being solved at row length 5, while 2 out of 9 can be solved with row length 7. The table also confirms that the use of a good initial solution is helpful – in 3 cases a larger instance with an initial solution is solved faster than a smaller instance with no initial solution.

B.7 Conclusions

A number of different modelling approaches for the solution of the Double TSP with Multiple Stacks have been proposed and tested. The problem has turned out to be very difficult to solve, confirming the results known from the literature for problems that combine routing and loading issues in the same optimization problem.

The most successful approach is based on a new model solved through a decomposition approach. All the 5 tested instances with 4 rows of length 4, with a total number of nodes of 34, were solved to optimality within an hour as were 4 of the 5 tested instances with 4 rows of length 5 (42 nodes).

Interestingly, it could be observed that the difficulty of a given problem instance depends not so much on the number of orders, as on the length of the rows. Thus, instances with 16 orders are easily solved when using 4 loading rows, while they are currently impossible to solve to optimality when using only 2 rows.

The availability of optimal solutions allowed us to check the quality of the

instance	n	N	$R \times L$	UB	LB	gap	time/s.
a280	20	9	$3\times3^{+}$	585	585	0%	2
	24	11	3×4	654	654	0%	51
	28	13	3×5	696	696	0%	19
	32	15	$3\times5^{+}$	792	792	0%	31
	36	17	3×6	945	945	0%	2277
	40	19	3×7	-	1017	-	3600
	44	21	$3\times7^{+}$	1127	1091	3.21%	3600
	48	23	3×8	-	11605	-	3600
	52	25	3×9	-	1192	-	3600
att532	20	9	$3\times3^{+}$	5361	5361	0%	2
	24	11	3×4	6399	6399	0%	23
	28	13	3×5	7261	7261	0%	102
	32	15	$3\times5^{+}$	7562	7562	0%	320
	36	17	3×6	11369	7737	31.95%	3600
	40	19	3×7	11413	7972	30.16%	3600
	44	21	$3\times7^{+}$	13218	12230	7.48%	3600
	48	23	3×8	-	12530	-	3600
	52	25	3×9	-	15709	-	3600
brd14051	20	9	$3\times3^{+}$	7897	7897	0%	0
	24	11	3×4	8064	8064	0%	1
	28	13	3×5	8079	8079	0%	41
	32	15	$3\times5^{+}$	8196	8196	0%	3
	36	17	3×6	8300	8226	0.89%	3600
	40	19	3×7	8434	8394	0.48%	3600
	44	21	$3\times7^{+}$	9109	8400	7.79%	3600
	48	23	3×8	-	8499	-	3600
	52	25	3×9	-	8513	-	3600
d15112	20	9	$3\times3^{+}$	93597	93597	0%	28
	24	11	3×4	100489	100489	0%	39
	28	13	3×5	108574	108574	0%	211
	32	15	$3\times5^{+}$	130297	124692	4.30%	3600
	36	17	3×6	141408	126627	10.45%	3600
	40	19	3×7	-	130153	-	3600
	44	21	$3\times7^{+}$	188222	132034	29.85%	3600
	48	23	3×8	-	133448	-	3600
	52	25	3×9	-	138886	-	3600
d18512	20	9	$3\times3^{+}$	7951	7951	0%	1
	24	11	3×4	8023	8023	0%	1
	28	13	3×5	8034	8034	0%	6
	32	15	$3\times5^{+}$	8098	8098	0%	19
	36	17	3×6	8567	8124	5.17%	3600
	40	19	3×7	-	8292	-	3600
	44	21	$3\times7^{+}$	10664	8425	21.00%	3600
	48	23	3×8	-	8476	-	3600
	52	25	3×9	-	8556	-	3600
fn14461	20	9	$3\times3^{+}$	3387	3387	0%	1
	24	11	3×4	3430	3430	0%	9
	28	13	3×5	3628	3628	0%	185
	32	15	$3\times5^{+}$	3796	3796	0%	192
	36	17	3×6	3853	3837	0.42%	3600
	40	19	3×7	5344	3981	25.52%	3600
	44	21	$3\times7^{+}$	4589	4058	11.58%	3600
	48	23	3×8	-	4170	-	3600
	52	25	3×9	-	4253	-	3600
nrw1379	20	9	$3\times3^{+}$	4572	4572	0%	3
	24	11	3×4	4733	4733	0%	17
	28	13	3×5	4872	4872	0%	273
	32	15	$3\times5^{+}$	4984	4984	0%	1230
	36	17	3×6	5355	5195	2.99%	3600
	40	19	3×7	-	5245	-	3600
	44	21	$3\times7^{+}$	6114	5434	11.12%	3600
	48	23	3×8	-	5481	-	3600
	52	25	3×9	-	5862	-	-
pr1002	20	9	$3\times3^{+}$	21498	21498	0%	0
	24	11	3×4	22977	22977	0%	15
	28	13	3×5	25087	25087	0%	184
	32	15	$3\times5^{+}$	25899	25899	0%	929
	36	17	3×6	27246	27246	0%	731
	40	19	3×7	28196	28196	0%	1733
	44	21	$3\times7^{+}$	29875	29875	0%	5
	48	23	3×8	31463	31463	0%	133
	52	25	3×9	32319	32319	0%	5
ts225	20	9	$3\times3^{+}$	34000	34000	0%	0
	24	11	3×4	43000	43000	0%	443
	28	13	3×5	48440	48440	0%	2
	32	15	$3\times5^{+}$	50580	50580	0%	4
	36	17	3×6	50881	50881	0%	2
	40	19	3×7	51371	51371	0%	17
	44	21	$3\times7^{+}$	52322	52322	0%	8
	48	23	3×8	54460	54460	0%	6
	52	25	3×9	62688	62688	0%	808

Table B.8: Results on instances from [11].

solutions produced by the heuristic presented in [91] for the DTSPMS. The heuristic solution turned out to always match the optimum for instances with 2 rows (14 optimal solutions known), often for 3 rows (9 out of 12 optimal solutions known), but rarely for 4 rows (1 out of 10 optimal solutions known).

B.8 Acknowledgements

The authors wish to acknowledge the anonymous referees whose suggestions helped to improve a previous version of this paper.

Appendix

This section will describe how the numbers of feasible solutions to a DTSPMS shown in Table B.1 are calculated.

The calculations will be done by first determining the number of feasible pickup routes, then the number of feasible loading plans given a pickup route, and finally the number of feasible delivery routes, given a loading plan. The calculations can naturally be reversed, and will be identical if starting with a delivery route and ending with the pickup route, or, as it turns out, if starting with a loading plan and then determining the number of each of the routes. A *loading plan* is a plan that gives the exact position of each loaded item, i.e., both its loading row and its position in that row, as opposed to a *row assignment*, which assigns each item to a row, but does not include the row position.

The number of feasible pickup routes is straightforward: N orders can be ordered in $N!$ different ways when no restrictions are present.

Next, the number of feasible loading plans for a given pickup route must be determined. In other words, given the arrival order of N items, determine the number of feasible loadings λ_N, respecting the number of rows R and the length of the rows L. With infinite row capacity, which in this case is equivalent to $L = N$, the number of feasible loading plans would be trivially R^N, since each item could be placed in any of the R rows. However, this is rarely the case in practice. Often $N = R \times L$, and the remainder of this section will assume that $L = \lceil \frac{N}{R} \rceil$.

If the loaded vehicle contains no empty space, i.e., if $N = R \times L$ the number of loading plans can be obtained as a product of one or more binomial coefficients:

$$\lambda_N = \prod_{r=0}^{R-2} \binom{N - r \cdot L}{L}. \tag{B.43}$$

This expresses the number of ways the orders can be grouped to fit the rows. Once L items have been assigned to a given row, the internal

positions within the row are given by the pickup routing. Thus, the number of feasible loading plans can be determined as the number of row assignments.

When the vehicle is not fully loaded the calculation is less simple, and the number of loading plans must be determined by recursion:

$$\lambda_m = \lambda_{m-1} \cdot \sum_{r=1}^{R} (r \cdot p_m^r), \text{ for } L < m \leq N \tag{B.44}$$

$$\lambda_m = R^m, \text{ for } m \leq L, \tag{B.45}$$

where p_m^r is the probability that there are r different rows to choose from, on the arrival of item m.

For the first $m \leq L$ loaded items, it is not yet possible that any row has been filled up (the $m-1$ items already loaded are too few to fill an entire row), so the first m items can each be placed in any of the R rows, and the total number of ways to divide the first m items into the R rows is R^m, as stated in (B.45).

For values of $m > L$, the value of λ_m can be obtained by determining, for each possible $r = 1, \ldots, R$, the probability that there are r rows available, and multiplying this by λ_{m-1}. The calculation of these probabilities p_m^r depends on the number of rows R and is described here for $R = 2$ and $R = 3$.

R=2 The probability that only one row is available is equal to the probability that one row is full. With $m-1$ items already loaded, there are $\binom{m-1}{L}$ different ways of selecting the L items to fill this row. The full row can then be either of the two existing rows and we obtain:

$$p_m^1 = \frac{\binom{m-1}{L} \cdot 2}{\lambda_{m-1}}. \tag{B.46}$$

Since there is necessarily either one or two rows available, it follows that $p_m^2 = 1 - p_m^1$.

R=3 The situation where only one row is available can only occur if $m > 2L$ (otherwise, there are not enough items already loaded to fill up two rows).

The L items that fill up the first full row can be chosen among the $m-1$ already loaded items in $\binom{m-1}{L}$ ways, while the L items to fill up the second row can be chosen among the remaining loaded items in $\binom{m-1-L}{L}$ ways. Then these two rows can be placed in 6 different ways among the 3 existing rows and we obtain:

$$p_m^1 = \frac{\binom{m-1}{L} \cdot \binom{m-1-L}{L} \cdot 6}{\lambda_{m-1}}. \tag{B.47}$$

The probability that there are two rows available for item m corresponds to the number of ways there can be exactly one row that is already full. First the items that complete the full row can be chosen in $\binom{m-1}{L}$ ways, similar to the 2 row case, and this full row can be either of the three existing ones. Next the distribution of the remaining $m-1-L$ customers between the remaining two rows is found, by summing over all possible lengths of one of these rows. Since we know that neither of these rows can be full, each must have at most $L-1$ items. In the sum j is the number of items in one of the two rows. Since these two rows are considered "symmetrically" this sum is not multiplied by two (as in the previous cases). We obtain:

$$p_m^2 = \frac{\binom{m-1}{L} \cdot 3 \cdot \sum_{j=\max(m-2L,0)}^{\min(m-1-L,L-1)} \binom{m-1-L}{j}}{\lambda_{m-1}}. \tag{B.48}$$

Finally, the probability that all three rows are still available can be found in the simple way:

$$p_m^1 + p_m^2 + p_m^3 = 1 \Leftrightarrow p_m^3 = 1 - p_m^1 - p_m^2. \tag{B.49}$$

If a loading plan is given, the number of feasible delivery routings can also be found by applying the recursion given in (B.44)–(B.45), by a similar argument.

For each of the first L visits, the next in line can be chosen among the R items available for delivery (at the end of each row). For visit number $L+1$ there is a probability p_{L+1}^{R-1} that one row is empty and therefore only $R-1$ rows contain items that can be chosen for the next delivery, and a similar argument can be applied to the remaining visits.

Now the total number of feasible solutions to a problem of a given size can be determined. First the pickup route is constructed. There are $N!$ feasible routes, when all loading constraints are disregarded. Next a corresponding loading plan can be constructed in λ_N different ways, and finally the delivery route can be constructed in another λ_N ways, leading to a total number of feasible solutions to the problem, of

$$K_N = N! \cdot (\lambda_N)^2. \tag{B.50}$$

Finally, it should be noted that if the distances are symmetric, each solution will lead to another solution of the same cost, by reversing each of the graphs.

In addition to the information provided by Table 1, in Table B.9 we show the number of feasible solutions to the DTSPMS for different problem instances with 2 and 3 rows. As the table shows the number of feasible solutions to an instance increases drastically with the number of orders, showing that simple complete enumeration is far from being a viable solution method even on small instances.

N	DTSP2S	DTSP3S
5	$4.8 \cdot 10^{4}$	$9.7 \cdot 10^{5}$
10	$2.3 \cdot 10^{11}$	$1.8 \cdot 10^{15}$
12	$4.1 \cdot 10^{14}$	$5.8 \cdot 10^{17}$
15	$2.2 \cdot 10^{20}$	$7.5 \cdot 10^{23}$
20	$8.3 \cdot 10^{28}$	$3.9 \cdot 10^{35}$
25	$1.7 \cdot 10^{39}$	$3.4 \cdot 10^{47}$
30	$6.4 \cdot 10^{48}$	$8.2 \cdot 10^{57}$
33	$4.7 \cdot 10^{55}$	$1.6 \cdot 10^{65}$

Table B.9: Number of feasible solutions with N orders.

Appendix C

The Simultaneous Vehicle Scheduling and Passenger Service Problem

Authors:
Hanne L. Petersen, Allan Larsen, Oli B.G. Madsen, Bjørn Petersen, Stefan Ropke.

Abstract:
Passengers using public transport systems often experience waiting times when transferring between two scheduled services. In this paper we propose a planning approach which seeks to obtain a favourable trade-off between the two contrasting objectives passenger service and operating cost by modifying the timetable. This planning approach is referred to as the Simultaneous Vehicle Scheduling and Passenger Service Problem (SVSPSP). The SVSPSP is modelled as an integer programming problem, and solved using a large neighborhood search (LNS) metaheuristic. The proposed framework is tested on data inspired by the express-bus network in the Greater Copenhagen Area. The results are encouraging and indicate a potential decrease of passenger waiting times in the network of 10–20%, with the vehicle scheduling costs remaining mostly unaffected.

C.1 Introduction

In every larger public transport system massive amounts of time are wasted due to waiting time when transferring between different parts of the journey. For the Greater Copenhagen area it has been estimated that the time lost on an average weekday by passengers waiting for connecting buses or trains approaches 65,000 hours (based on 400,000 daily transfers with an average of 10 minutes transfer waiting time.[1] Hence, generating timetables which optimise for temporal correspondences has an enormous socio-economic potential. Clearly, this could be achieved through an increase in the frequency of the trips offered in the timetable, however this would require an unacceptable increase in operating costs.

The traditional sequential framework for planning of public transport has been excellently described by Desaulniers and Hickman [26] and is sketched in Figure C.1. Given the route network, the frequencies are determined to ensure demand coverage and to comply with politically determined service levels, under practical constraints such as fleet size. The timetabling process then determines the exact timings for all trips while respecting the previously determined frequencies/headways. Both of these first phases are concerned with maximising some measure of passenger service, and are carried out by the public transport service provider, who typically works by appointment by the local authorities. The timetabling phase may take schedule synchronisation and transfer times into account.

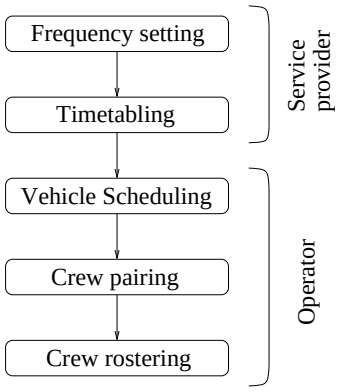


Figure C.1: Traditional sequential planning approach.

Once the timetable has been established, the resource scheduling begins. During this phase the first problem to be solved is the scheduling of the physical resources necessary to carry out the trips in the timetable, i.e. the vehicles. The purpose of the vehicle scheduling is to be able to execute the timetable at the lowest possible cost. The costs considered in this phase include empty mileage performed by the vehicles, both in connection to the depot, and in the form of *deadheading*, i.e. transport

[1] cf. http://www.dtu.dk/centre/modelcenter/TU/Standard%20Tabeller/

between the end point of one trip and the starting point of another. Once the vehicle schedules have been established, the crew pairing and rostering phases are carried out. The last three phases are all carried out by the public transport operator, who is appointed by the service provider to operate a set of trips, and they all have the purpose of operating the requested timetable at the lowest possible cost.

Today, efficient systems for generating near-optimal vehicle schedules exist within all modes of transport. However, these systems treat the timetable as fixed input, meaning that potential savings in operating costs from moving a set of trips in the timetable are lost. Only very limited research has been done on models that address the problem of minimising the operating costs by modifying the timetable. Furthermore, research is scarce on models that focus on minimisation of the waiting time during transfer.

In this paper we introduce the Simultaneous Vehicle Scheduling and Passenger Service Problem (SVSPSP) which addresses the multiple objective planning problem of improving timetables such that they remain economically satisfactory for the operator, and at the same time offer high-quality service to the passengers by reducing the unproductive time spent on waiting during transfers. Please note that whenever we refer to *waiting time* throughout this paper we are solely referring to the waiting time associated with transfers, and not the waiting time of passengers entering the system. The SVSPSP framework is sketched in Figure C.2, and integrates the planning processes of timetabling and vehicle scheduling.

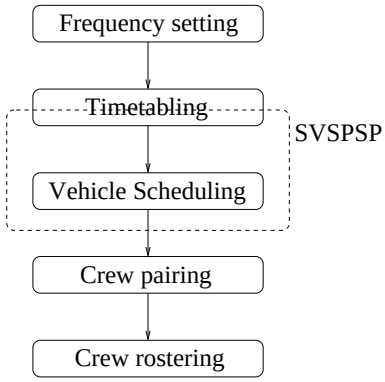


Figure C.2: The role of the SVSPSP shown in relation to the traditional sequential planning approach.

The main input of the SVSPSP is the original timetable and estimates of passenger demand in the network. The natural problem owner of the SVSPSP is the public transport service provider, as this is the authorithy which on the one hand is committed to provide a high-quality timetable to the customers (in terms of e.g. minimum waiting times) and on the other hand holds the responsibility of ensuring that the offered timetable

is feasible from an operating costs perspective. By integration of the vehicle scheduling phase, which previously belonged to the operator, the service provider can obtain a better negotiating position towards the operator, since the operating costs have already been considered during the optimisation of the timetables.

The contributions of this paper are fourfold: 1) we formally introduce a new interesting problem, motivated by a real-life case, 2) we make a realistic data set available, that can be used for future studies, 3) we propose a heuristic solution method that is able to handle data sets of realistic size, 4) we show that substantial reductions in passenger waiting time are possible using the proposed methodology. The paper is organised as follows: Section C.2 reviews the literature on the multiple depot vehicle scheduling problem as well as work on minimising passenger transfer times. In section C.3 we formulate the SVSPSP as an integer programming model. Section C.4 discusses how the proposed problem can be solved by the large neighborhood search metaheuristic. Section C.5 introduces the data set used in this study which is based on the bus network of the Greater Copenhagen area, and in Section C.6 we discuss the results obtained. Finally, we provide our concluding remarks and suggest directions for further research in Section C.7.

C.2 Literature review

Our approach to the integrated vehicle scheduling and timetabling problem is based on the *multiple depot vehicle scheduling problem* (MDVSP). Desrosiers et al. [29] provide an excellent introduction to the problem and survey the literature prior to 1995. A more recent, but short literature survey is presented by Pepin et al. [89] who also present an interesting comparison of heuristic approaches for the problem. Section 4.1 in Desaulniers and Hickman [26] also contains a recent survey. Some of the currently best exact methods for the MDVSP are proposed by Hadjar et al. [56] and Löbel [75]. We are aware of two papers that extend vehicle scheduling problems to handle parts of the timetabling process. The paper by van den Heuvel et al. [60] studies the integration of timetabling and multi depot vehicle scheduling with the aim of reducing costs (reducing the number of vehicles) while ignoring passenger waiting times. On the timetabling level the approach allows the trip starting times for each line to be shifted in time to allow greater flexibility in the vehicle scheduling part. The paper presents integer programming models as well as a local search algorithm that solves a network flow problem in each local search iteration. Guihaire and Hao [53] also integrate vehicle scheduling and timetable synchronisation in their optimisation problem. They consider several terms in their objective: number of vehicles required, number and quality of transfer possibilities and the so-called headway evenness. The second term aims at minimising passenger inconvenience. The last term attempts to make arrivals of vehicles, serving a particular

line, occur with a regular frequency. The three terms are weighted together. In terms of the vehicle scheduling problem, the paper considers a single depot setup while our approach handles the multiple depot case. The problem studied in this paper is probably the one that resembles our problem the most.

Several papers focus on optimising timetables in order to minimise passenger waiting times, without explicitly considering the impact such changes have on the physical resource requirements (e.g. more buses may be needed to carry out the modified plan). Examples of such approaches are Jansen and Pedersen [62] who formulate the problem as a mathematical model and propose simulated annealing and tabu search algorithms to solve the problem (see also Pedersen [87]); Ceder et al. [14] who synchronise bus timetables by maximising the number of times two buses arrive at the same time at any node in the network; Klemt and Stemme [67], Bookbinder and Désilets [8] and Daduna and Voß [25] who synchronise timetables by solving a quadratic semi-assignment problem. Worth mentioning is also the paper by Chakroborty et al. [16], which studies timetable synchronisation and "optimal fleet size" using a genetic algorithm heuristic. They do not study the vehicle scheduling aspect of the problem, instead the term "optimal fleet size" refers to the fact that the number of departures on a specific line is a variable, decided by the proposed model.

As explained in the introduction, the SVSPSP integrates the timetabling and vehicle scheduling phases. The integrated problem has not been widely studied in the literature but some papers on the topic do exist. One approach for handling the integrated problem has been the so-called *periodic event scheduling problem* (PESP). The PESP is mainly used for timetabling but has been extended to handle some aspects of vehicle scheduling as well. The PESP model was proposed by Serafini and Ukovich [102]. It is a general framework for modelling optimisation problems with a periodic nature. Liebchen and Möhring [73] show how the PESP and extensions can be used to handle many aspects of railway timetabling. One of these is to minimise the changeover time for passengers and another is the minimisation of the number of vehicles needed to perform the timetable. The complexity of the vehicle minimisation depends on whether trains are allowed to switch line when they reach their endpoint. Contrary to our approach the paper does not model the situation where vehicles can perform deadheading in order to switch terminal (this does not seem practical when the vehicles are trains running on tracks, but can be useful for buses). The material in Liebchen and Möhring [73] builds on the work of Liebchen and Peeters [74] which focuses on vehicle minimisation, but arrives at a model with a quadratic objective function. Other recent works on the PESP and railway timetabling include Liebchen and Möhring [72], Peeters [88], and Kroon et al. [68].

Wong et al. [113] study the *Mass Transit Railway* in Hong Kong that

contains 6 train lines. They minimise the overall passenger waiting time in a non-periodic fashion. The number of vehicles needed to carry out the plan is determined in advance and is kept constant. In this way it is ensured that the proposed timetable does not become too expensive to carry out, while optimising customer satisfaction. The authors present a MIP model and solve it using a heuristic that incorporates a standard MIP solver as an important component. Fleurent et al. [37] describe an optimisation system and an interactive tool for minimising passenger waiting time while keeping vehicle costs under control. The suggested approach is tested on a case from the city of Montreal, Canada, and the results indicate that the passenger waiting time can be improved while keeping the vehicle count constant. The paper provides little detail about the optimisation algorithm used to obtain these results.

We can conclude that the work on integrating time tabling and vehicle scheduling is rather limited and that Guihaire and Hao [53] is the paper that presents a problem that is most similar to the SVSPSP. The SVSPSP model is, regarding some aspects, more ambitious than the model studied by Guihaire and Hao [53] as it considers a multi-depot setting which is not the case in the aforementioned paper.

C.3 The SVSPSP: modelling

In a classical multi-depot vehicle scheduling problem (MDVSP) a set of trips have to be covered with a set of vehicles (based at several depots) while minimising costs. A *trip* has a start and end location, as well as a departure and arrival time. In a bus scheduling setting a trip corresponds to the movement from the start to the end of a bus line. A *line* is a collection of trips that have the same start and end locations but different departure and arrival times. A line also contains trips going in the opposite direction. The MDVSP can be modelled as follows (see [29]): let $N = \{1, \ldots, n\}$ denote the set of trips and K the set of depots. With each depot $k \in K$ we associate a graph $G^k = (V^k, A^k)$ where the set of nodes is defined as $V^k = N \cup \{n+k\}$ with $n+k$ being the node representing the k^{th} depot. The set of arcs A^k is a subset of the set $V^k \times V^k$, with all infeasible arcs removed. An arc is infeasible if it forms an impossible connection between two trips; typically this is caused by timing constraints. For each depot $k \in K$ and each arc $(i, j) \in A^k$ we define an arc cost c_{ij}^k and we are given an upper bound v^k on the number of vehicles located at k. Using a binary variable x_{ij}^k for all $k \in K, (i, j) \in A^k$, having value 1 if and only if a vehicle from depot k travels from node i to j we can write an integer multi-commodity flow model as follows:

$$\min \sum_{k \in K} \sum_{(i,j) \in A^k} c_{ij}^k x_{ij}^k \tag{C.1}$$

subject to

$$\sum_{k\in K}\sum_{j\in V^k} x_{ij}^k = 1 \qquad i \in N \tag{C.2}$$

$$\sum_{j\in N} x_{n+k,j}^k \leq v^k \qquad k \in K \tag{C.3}$$

$$\sum_{i\in V^k} x_{ij}^k - \sum_{i\in V^k} x_{ji}^k = 0 \qquad k \in K, j \in V^k \tag{C.4}$$

$$x_{ij}^k \in \{0,1\} \quad k \in K, (i,j) \in A^k. \tag{C.5}$$

The objective (C.1) minimises the total cost. The arc costs c_{ij}^k can be set such that the total cost reflects a fixed cost per vehicle and deadheading costs. Constraints (C.2) ensure that all trips are served, constraints (C.3) ensure that we do not use more than the available number of vehicles, and constraints (C.4) are flow conservation constraints.

The SVSPSP generalises the MDVSP as follows: in the SVSPSP we group trips into so called *metatrips*. The set of metatrips, Ω, forms a partitioning of the set N, that is, $\cup_{M\in\Omega} M = N$ and $\forall M_1, M_2 \in \Omega, M_1 \neq M_2 : M_1 \cap M_2 = \emptyset$. Furthermore, we relax the condition that every trip must be covered. Instead we require that exactly one trip from each metatrip must be covered. In the context of this paper, we assume that each metatrip corresponds to a trip from the original timetable, and the (sub)trips belonging to the metatrip represent copies of the original trip, with alternative departure times. Thus, the requirement that each metatrip is covered corresponds to the MDVSP-requirement that each trip is covered (C.2). The idea behind this, in relation to our goal of increasing passenger service, is that selecting alternative departure times may reduce waiting times and thereby improve the passenger service level.

We will now introduce some useful concepts that will be used in our treatment of the SVSPSP. Trips in the SVSPSP model can be *incompatible* for various reasons, as we shall see later. This is captured by a set $\Phi \subseteq 2^N$ containing sets of mutually incompatible trips. Thus, if $\phi \in \Phi$ then any pair $i, j \in \phi$ is incompatible and cannot be used together in a feasible solution. For the SVSPSP we maintain the definition of a *line* that is known from the MDVSP; a line L is a sequence of *stops* to be visited in a given order. A line can be travelled in both directions, and we use the term d-line (directed line) for a line in a particular direction. Each metatrip, and the trips contained in it, belongs to exactly one d-line. Therefore we can view a d-line L as a subset of the set of metatrips: $L \subseteq \Omega$. For every bus line a number a *stops* are defined. The stops are the locations where the bus stops to pick up and unload passengers. Several bus lines may share one stop and a stop can provide connections to other modes of timetabled transportation like trains or ferries. Any transfer of passengers takes place at a stop. We are only interested in stops where transfers can take place, hence, when mention-

ing stops in the rest of this paper we assume a stop with at least one transfer opportunity.

Figure C.3 shows an example of trips and metatrips. The nodes $\{1, \ldots, 12\}$ represent trips, and two metatrips $\{2, \ldots, 6\}$ and $\{7, \ldots, 11\}$ are shown. The time of day is shown along the top of the figure. Trips 4 and 9, marked with grey, are the two original trips, from which the metatrips are constructed. The remaining trips in each metatrip are constructed by creating duplicates of the original trip, spread evenly in the available time interval. The nodes 1 and 12 belong to other metatrips, not illustrated in the figure. All trips shown in the figure belong to the same d-line.

The usage of incompatible trips to impose passenger service is apparent: trips belonging to the same d-line and departing within a short time interval should be incompatible, for example trip 6 and 7 on Figure C.3 could be incompatible because they depart within 3 minutes. Similarly, two consecutive departures on a d-line should not be too far apart. Therefore it would make sense to make trip 2 incompatible with trip 11. If departures at regular intervals are required on a bus line for a specific period of the day or the entire day this could also be modelled using incompatible trips. If we desire departures every 15 minutes in the example on Figure C.3 we must make trip 2 incompatible with trips 8, 9, 10, and 11 (by adding the set $\{2, 8, 9, 10, 11\}$ to Φ), trip 3 should be incompatible with trips 7, 9, 10, and 11, and so on.

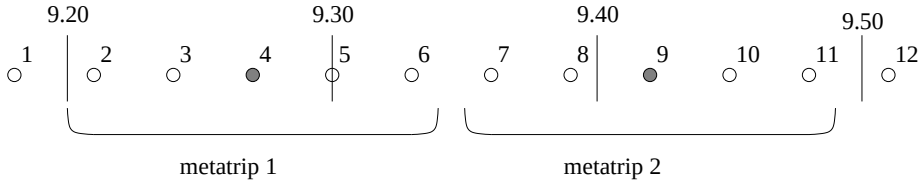


Figure C.3: Example of trips and metatrips.

Using the notation from the MDVSP we can now present a mathematical model for a simple version of the SVSPSP, denoted SVSPSP^0.

$$\min \sum_{k \in K} \sum_{(i,j) \in A^k} c_{ij}^k x_{ij}^k \tag{C.6}$$

subject to

$$\sum_{i \in M} \sum_{k \in K} \sum_{j \in V^k} x_{ij}^k = 1 \qquad M \in \Omega \tag{C.7}$$

$$\sum_{i \in \phi} \sum_{k \in K} \sum_{j \in V^k} x_{ij}^k \leq 1 \qquad \phi \in \Phi \tag{C.8}$$

$$\sum_{j \in N} x_{n+k,j}^k \leq v^k \qquad k \in K \tag{C.9}$$

$$\sum_{i \in V^k} x_{ij}^k - \sum_{i \in V^k} x_{ji}^k = 0 \qquad k \in K, j \in V^k \tag{C.10}$$

$$x_{ij}^k \in \{0,1\} \quad k \in K, (i,j) \in A^k. \tag{C.11}$$

Constraints (C.9) and (C.10) are identical to (C.3) and (C.4) in the original MDVSP formulation. Constraints (C.7) ensure that exactly one trip from each metatrip is selected and constraints (C.8) ensure that no pair of incompatible trips are selected at the same time.

In order to discuss how passenger service can be taken into account in the SVSPSP^0 we need to define exactly how we measure passenger service. The area we focus on in relation to passenger service is waiting time during transfers. We first introduce the central concept *transfer opportunity*. A transfer opportunity is a triple (s, M, L), where s is the stop where the transfer takes place, M is a metatrip that stops at s, and L is a connecting line that exchanges passengers with M at s. For each transfer opportunity we assume that an estimate D_{ML}^s of the number of passengers disembarking metatrip M and transferring to line L at stop s, as well as an estimate E_{ML}^s of the number of passengers embarking metatrip M transferring from line L at stop s are available. It is assumed that all passengers disembarking a metatrip to transfer to line L take the earliest possible departure on line L and all passengers embarking a metatrip M come from the latest possible arrival on line L. For the SVSPSP^0, L is a line external to the model, but we will later generalise it to include those lines that are rescheduled by the model.

To improve passenger service we desire to minimise the total number of passenger minutes wasted by waiting for a connection, at the same time as we want to minimise the cost of serving all trips. This results in two goals that are weighted together in the cost coefficients of the objective function. The SVSPSP^0 model can accommodate a part of the waiting times that we desire to include in the model, namely a penalty for waiting times related to lines that are external to the model, such as already timetabled train departures: for each trip i in N we find the transfer opportunities (s, M, L) of the metatrip M that i belongs to. As stated above, L is an external line with fixed departures and arrivals, therefore we can a priori find the arrival and departure on line L that are used by passengers embarking and disembarking trip i at stop s and we can calculate the associated waiting times. The two waiting times are multiplied by the passenger estimates E_{SM}^s and D_{SM}^s and summed to give

the total number of minutes waited for the particular trip and transfer opportunity. By summing over all the transfer opportunities that the trip is involved in we obtain the total number of waiting minutes incurred by the trip. This number, weighted in a suitable way, is added to the cost of all arcs leaving the node corresponding to the trip.

The SVSPSP^0 model cannot take the transfer of passengers from bus to bus into account if both buses are rescheduled by the model. We therefore introduce the model SVSPSP, that generalises SVSPSP^0, to accommodate this. The overall idea is to introduce two new sets of binary variables y^s_{ij} and z^s_{ij} that indicate if transfers between trip i and j are taking place at stop s. For each transfer opportunity (s, M, L) involving a d-line L which is timetabled by the model we create a number of variables y^s_{ij} where $i \in M$, $j \in \cup_{M' \in L} M'$. Each variable indicates if the transfer opportunity of passengers disembarking metatrip M to transfer to d-line L is realised by transferring from trip i to j. Similarly, for the same transfer opportunity, we create a number of variables z^s_{ij} where $j \in M$, $i \in \cup_{M' \in L} M'$. These variables indicate if the transfer opportunity of passengers embarking M, coming from L is realised by transferring from trip i to j. We assign a cost $\bar{c}^s_{ij} > 0$ for each y^s_{ij} variable and a cost $\hat{c}^s_{ij} > 0$ for each z^s_{ij} variable. The cost is based on the time between arrival and departure on the two trips and the number of passengers expected to take advantage of the transfer opportunity.

Consider the following example: the bus lines 200 and 300 both visit *Lyngby* station. Assume that a trip for line 200 northbound (200-N) has been chosen by the model such that the bus arrives at Lyngby station at 9.29. A number of the passengers on board the bus wish to disembark the bus to transfer to line 300 heading north (300-N). Their waiting time depends on the departure time of the next 300-N, which is also decided by the model. Figure C.4 shows this situation. The chosen

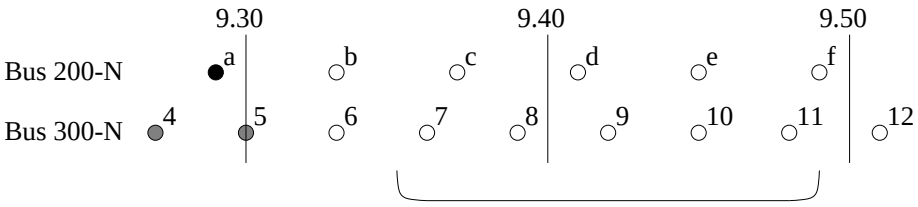


Figure C.4: Example of a bus-to-bus transfer.

trip for bus 200-N (trip `a`) is shown at the top of the figure along with alternative 200-N arrivals, and nine trips belonging to line 300-N are shown on the bottom. Passengers from trip `a` cannot transfer to bus 300-N on the departure times marked with grey circles: departure 4 is impossible because it departs before bus 200-N arrives, while departure 5 departs one minute later than trip `a` arrives and there is not enough time for the transfer (passengers have to walk). The other departures are all feasible transfers. Note that trips 7 to 11 constitute a metatrip, so exactly one of these trips must be selected. This means that no passenger from

trip a heading for line 300-N would transfer to trip 12 because an earlier, feasible departure will exist in the plan. On the other hand, if trip 12 is selected by the model and trip a is the latest selected bus from 200-N that allows a transfer to trip 12 then embarking passengers on trip 12 arriving from 200-N would perform the transfer. Since both embarking and disembarking passengers are considered, both y and z variables are necessary. The y variables handle passengers *disembarking* a specific trip to the first possible trip on the specified d-line. The z variables handle passengers *embarking* a specific trip from the last possible trip on the specified d-line.

Let S be the set of all stops that are visited by more than one bus line. We introduce a graph $\hat{G}^s = (\hat{V}^s, \hat{A}^s)$ for each stop $s \in S$. The set of vertices $\hat{V}^s$ is the set of all trips that visit stop s and the set of arcs is defined as

$$\hat{A}^s = \left\{ (i,j) : i,j \in \hat{V}^s, \begin{array}{l} \text{passengers can transfer from trip } i \text{ to} \\ \text{trip } j \text{ at stop } s \end{array} \right\}.$$

For example, if s is Lyngby station as shown in Figure C.4 we would have that

$$\{(a,6),(a,7),(a,8),(a,9),(a,10),(a,11),(a,12)\} \subset \hat{A}^s,$$

but $\{(b,4),(b,5),(b,6)\} \cap \hat{A}^s = \emptyset$. The variables y^s_{ij} and z^s_{ij} are defined for every $s \in S$ and every arc $(i,j) \in \hat{A}^s$. We can use Figure C.4 to show the meaning of the y variables. If, for example, trips a and 9 are chosen and trip 6 is not then $y^s_{\mathtt{a},9} = 1$ and $y^s_{\mathtt{a},j} = 0$ for $j \in \{6,7,8,10,11,12\}$. If both trip 6 and 9 were chosen then we would have $y^s_{\mathtt{a},6} = 1$ and $y^s_{\mathtt{a},9} = 0$ because all passengers disembarking a, bound for 300-N, would transfer to trip 6.

For a trip $i \in N$ and a stop $s \in S$ on its line we define $t(i,s)$ to be the departure time of trip i at stop s. For a trip i we define $dl(i)$ to be the d-line that the trip belongs to. For a stop s and an arc $(i,j) \in \hat{A}^s$ we define

$$\pi(i,j,s) = \{j' \in \cup_{M' \in dl(j)} M' : (i,j') \in \hat{A}^s, t(j') < t(j)\},$$

that is, $\pi(i,j,s)$ is the set of trips j' from the same d-line as j that are earlier than j but that still are feasible transfer destinations from trip i. Similarly we define

$$\sigma(i,j,s) = \{i' \in \cup_{M' \in dl(i)} M' : (i',j) \in \hat{A}^s, t(i) < t(i')\},$$

which is the set of trips i' from the same d-line as i that are later than i but where a transfer to trip j still is feasible. We can now present an extended model that also handles the bus-to-bus transfers:

$$\min \sum_{k \in K} \sum_{(i,j) \in A^k} c^k_{ij} x^k_{ij} + \sum_{s \in S} \sum_{(i,j) \in \hat{A}^s} \bar{c}^s_{ij} y^s_{ij} + \sum_{s \in S} \sum_{(i,j) \in \hat{A}^s} \hat{c}^s_{ij} z^s_{ij} \qquad \text{(C.12)}$$

subject to

$$\sum_{i\in M}\sum_{k\in K}\sum_{j\in V^k} x_{ij}^k = 1 \qquad M \in \Omega \tag{C.13}$$

$$\sum_{i\in \phi}\sum_{k\in K}\sum_{j\in V^k} x_{ij}^k \leq 1 \qquad \phi \in \Phi \tag{C.14}$$

$$\sum_{j\in N} x_{n+k,j}^k \leq v^k \qquad k \in K \tag{C.15}$$

$$\sum_{i\in V^k} x_{ij}^k - \sum_{i\in V^k} x_{ji}^k = 0 \qquad k \in K, j \in V^k \tag{C.16}$$

$$\sum_{k\in K}\sum_{l\in V^k} x_{il}^k + \sum_{k\in K}\sum_{l\in V^k} x_{jl}^k - 1 - \sum_{j'\in\pi(i,j,s)}\sum_{k\in K}\sum_{l\in V^k} x_{j'l}^k \leq y_{ij}^s \qquad s \in S, (i,j) \in \hat{A}^s \tag{C.17}$$

$$\sum_{k\in K}\sum_{l\in V^k} x_{il}^k + \sum_{k\in K}\sum_{l\in V^k} x_{jl}^k - 1 - \sum_{i'\in\sigma(i,j,s)}\sum_{k\in K}\sum_{l\in V^k} x_{i'l}^k \leq z_{ij}^s \qquad s \in S, (i,j) \in \hat{A}^s \tag{C.18}$$

$$x_{ij}^k \in \{0,1\} \quad k \in K, (i,j) \in A^k \tag{C.19}$$

$$y_{ij}^s \in \{0,1\} \quad s \in S, (i,j) \in \hat{A}^s \tag{C.20}$$

$$z_{ij}^s \in \{0,1\} \quad s \in S, (i,j) \in \hat{A}^s. \tag{C.21}$$

Two changes have been performed compared to model SVSPSP^0: a) two terms have been added to the objective function (C.12) to model the cost of passengers waiting during transfers between two buses that are both re-timetabled by the model, and b) inequalities (C.17) and (C.18) have been added to ensure that the y_{ij}^s and z_{ij}^s variables are set correctly. For example, y_{ij}^s is set to 1 by (C.17) when both trip i and trip j are used (the first two sums on the left hand side) and when none of the feasible transfer destinations earlier than j are in use (the last sum on the left hand side). The constraints only enforce a lower bound on y_{ij}^s but the minimisation in the objective and assumption that $\bar{c}_{ij}^s$ is positive ensure that the y variables take the lowest possible value. Constraints (C.18) are similar to (C.17), but work on z rather than y variables.

The mathematical model presented in (C.6)–(C.11) has been implemented in CPLEX, but CPLEX was not able to solve instances with the dimensions considered in this paper. No attempts have been made to solve the model presented in (C.12)–(C.21) with a general purpose solver, since the number of variables and constraints used in the advanced model is even larger than in the model presented in (C.6)–(C.11). However, by presenting the models here, they have served as an instrument to give a precise definition of the problem to be studied. Using techniques like reformulation or cut or column generation it might be possible to solve realistically sized instances using the mathematical models – in particular, model (C.6)–(C.11) lends itself to a column based solution

approach. However, we have worked in a different direction, and in the following section we shall present a metaheuristic for solving the problem.

C.4 Solution method

The solution method we propose for solving the SVSPSP is based on the large neighborhood search (LNS) metaheuristic. The LNS was proposed by Shaw [104]. As many other metaheuristics, the LNS is based on the idea of finding improving solutions in the neighborhood of an existing solution. What sets the LNS apart from other metaheuristics is that the neighborhood searched (or sampled) in the LNS is huge.

The term LNS is often confused with the term *very large scale neighborhood search* (VLSN) defined by Ahuja et al. [1]. While the LNS is a heuristic framework, VLSN is the family of heuristics that searches neighborhoods whose sizes grow exponentially as a function of the problem size, or neighborhoods that simply are too large to be searched explicitly in practice, according to Ahuja et al. [1]. The LNS is one example of a VLSN heuristic.

We are aware of one application of LNS to the MDVSP, this approach is described by Pepin et al. [89]. That LNS implementation is more complex than ours as it uses column generation and branch and bound to solve restricted instances of the MDVSP. The computational results reported in [89] show that the LNS is competitive against four other heuristics. LNS has also been successful in solving the related vehicle routing problem with time windows. See for example Bent and van Hentenryck [5] and Pisinger and Røpke [93].

C.4.1 Large neighborhood search

An LNS heuristic moves from the current solution to a new, hopefully better, solution by first *destroying* the current solution and then *repairing* the destroyed solution. To illustrate this, consider the traveling salesman problem (TSP). In the TSP we are given n cities and a cost matrix that specifies the cost of traveling between each pair of cities. The goal of the TSP is to construct a minimum cost cycle that visits all cities exactly once (see e.g. [3]). A destroy method for the TSP could be to remove 10% of the cities in the current tour at random (shortcutting the tour where cities are removed). The repair method could insert the removed cities again using a cheapest insertion principle (see e.g. [63]).

The LNS heuristic is outlined on Algorithm 1. In the pseudo-code we use the symbols x for the current solution, x^* for the best solution observed during the search and x' for a temporary solution. The operator $d(\cdot)$ is the destroy method. When applied to a solution x it returns a partially destroyed solution. The operator $r(\cdot)$ is the repair method. It

Algorithm 1 Large Neighborhood Search

```
1: input: a feasible solution x;
2: x* = x;
3: repeat
4:    x' = r(d(x));
5:    if accept(x', x) then
6:       x = x';
7:    end if
8:    if f(x') < f(x*) then
9:       x* = x';
10:   end if
11: until stop criterion is met
12: return x*
```

can be applied to a partially destroyed solution and returns a normal solution. The expression $r(d(x))$ therefore returns a solution created by first destroying x and then rebuilding it.

The LNS heuristic takes an initial solution as input and makes it the current and best known solutions in lines 1 and 2. Lines 4 to 10 form the main body of the heuristic. In line 4 the current solution is first destroyed and then repaired, resulting in a new solution x'. In line 5 the new solution is evaluated to see if it should replace the current solution, this is done using the function *accept* which will be described in Section C.4.2.3 below. In lines 8 to 10 the best known solution is updated if necessary. Line 11 checks the stopping criterion which in our implementation simply amounts to checking if $t_{\max}$ seconds have elapsed.

C.4.2 Large neighborhood search applied to the SVSPSP

This section describes how the LNS heuristic has been tailored to solve the SVSPSP. In particular, we describe the implemented destroy and repair methods and the acceptance criterion.

C.4.2.1 Destroy methods

Destroy methods for the SVSPSP remove trips from the current solution. Every time a destroy method is invoked the number of trips to remove is selected randomly in the interval $[5, 30]$. Two simple destroy methods for the SVSPSP have been implemented. The first method simply remove trips at random, which is a good method for diversifying the search.

The second method is based on the *relatedness* principle proposed by [104]. Here we assign a relatedness measure $R(i, j)$ to each pair of trips (i, j). A high relatedness measure indicates that the two trips are highly

related. The relatedness of two trips i and j are defined as

$$\begin{aligned} R(i,j) =& 30 \times \mathbf{1}_{s(i)=s(j)} \; + \; 30 \times \mathbf{1}_{e(i)=e(j)} \; + \; 20 \times \mathbf{1}_{s(i)=e(j)} \\ & + \; 20 \times \mathbf{1}_{e(i)=s(j)} \; - \; |t(i) - t(j)| \end{aligned}$$

where $s(i)$ and $e(i)$ are the start and end locations of trip i respectively, $t(i)$ is the start time of trip i (start time in the current solution). The notation $\mathbf{1}_{expr}$ is used to represent the indicator function which evaluates to one if *expr* evaluates to true and zero otherwise. The measure defines two trips to be related if they start around the same time and if the share start and/or end locations. The measure is used to remove trips as follows. An initial seed trip is selected at random and added to a set of removed trips S. For each trip i still in the solution we calculate the relatedness

$$v(i,S) = \max_{j \in S} \{R(i,j)\}$$

The trips still in the solution are sorted according a non-increasing $v(i,S)$ in a sequence T, a random number p in the interval $[0,1)$ is drawn and the trip at position $\lfloor |T| p^5 \rfloor$ in T is selected. This selection rule favours trips with high $v(i,S)$ value. The selected trip is added to the set of removed trips, and $v(i,S)$ is recalculated after adding a trip to S. We continue to add trips to S, until we have reached the target number of removed trips.

The two destroy methods are mixed in the LNS heuristic. Before removing a trip from the solution it is decided which destroy method that should be used to select the trip. With probability 0.15 the first method (random) is used and with probability 0.85 the second method (relatedness) is used.

The trips that have been removed from the solution are still *active* in the sense that they will be used in the trip incompatibility check defined by constraints (C.8) and (C.14). That is, when adding a trip to a solution in the repair step below, we check if it is compatible with the trips in the solution and the trips removed in the previous destroy operation. A trip i is made inactive when another trip, belonging to the same metatrip as i, is inserted into the solution.

C.4.2.2 Repair methods

The repair method for the SVSPSP reinserts the trips that were removed from the solution by the destroy method. The repair method uses a randomised greedy heuristic. For each unassigned metatrip S the heuristic calculates an insertion cost $f(S)$ given the current solution. When inserting a metatrip S we have a choice of which trip $i \in S$ that should be inserted. With probability ρ we insert the same trip that was used in the solution before destruction and with probability $1 - \rho$ we insert a random trip from S. The chosen trip should be compatible with all active trips. Such a trip exists because we are sure that the trip from

the pre-destruction solution is still compatible with all trips. The requirement ensures that we never get to a situation where one or more metatrips cannot be inserted because of the the compatibility constraints (C.8) and (C.14).

Given the choice of trip i, we define the cost $f(S)$ as the cost of inserting trip i at the best possible position in the current solution multiplied by a random factor that is meant to diversify the insertion procedure. More precisely the cost is defined as:

$$f(S) = \begin{cases} \min_{r \in R}\{c(i,r)\} \cdot (1 + \text{rand}(-\delta, \delta)) & \text{if } \min_{r \in R} c(i,r) \neq \infty, \\ c(i, \emptyset) & \text{otherwise,} \end{cases}$$

where $c(i,r)$ is the cost of inserting trip i in route r at the best possible position, R is the set of routes in the current solution, $c(i,\emptyset)$ is the cost of serving the trip using a new vehicle from the best possible depot, δ is a parameter and $\text{rand}(-\delta,\delta)$ is a function that returns a random number in the interval $[-\delta,\delta]$. The parameter δ controls the amount of randomisation applied by the insertion procedure. The heuristic chooses to insert the metatrip S with lowest cost. It does this by inserting the trip i that was used as a representative for S and inserts this at its best possible position. This continues until all metatrips have been inserted. With the assumption that $v^k = |\Omega|$ it is always possible to insert a metatrip – we will always be able to serve it using a new vehicle.

C.4.2.3 Acceptance criterion

The acceptance criterion used in our implementation of the LNS heuristic is the one used in simulated annealing metaheuristics: The function $\text{accept}(x', x)$ used in line 6 of Algorithm 1 accepts the new solution x' if it is at least as good as the current solution x, that is, $f(x') \leq f(x)$. If $f(x') > f(x)$ then the solution is accepted with probability

$$e^{\frac{f(x)-f(x')}{T}}.$$

The parameter T is called the *temperature* and controls the acceptance probability: a high temperature makes it more likely that worse solutions are accepted. Normally the temperature is reduced in every iteration using the formula $T_{\text{new}} = \alpha T_{\text{old}}$ where $0 < \alpha < 1$ is a parameter that is set relative to desired start and end temperatures and desired number of iterations. Because we use elapsed time as stopping criterion we calculate the current temperature by the formula

$$T(t) = T_s \cdot \left(\frac{T_e}{T_s}\right)^{\frac{t}{t_{\max}}}$$

here t is the elapsed time since the start of the heuristic, T_s is the starting temperature and T_e is the end temperature. Because of the acceptance criterion the LNS heuristic can be seen as a simulated annealing heuristic with a complex neighborhood definition.

Algorithm 2 Heuristic for generating an initial solution

```
1: while there are non-served metatrips left do
2:   Select a random station s with unserved metatrips;
3:   Select earliest non-served metatrip S starting from s;
4:   Start a new route r serving S. Use a vehicle from the depot nearest
     to s;
5:   repeat
6:     Let s' be the station where route r is ending;
7:     if r can be extended with a non-served metatrip starting in s'
       then
8:       Select earliest non-served metatrip S' starting in s' that can
         extend r. Add S' to r;
9:     else
10:      End route r by returning to the depot;
11:    end if
12:  until r has returned to the depot;
13: end while;
```

C.4.2.4 Starting solution

A starting solution is necessary because the LNS heuristic improves an existing solution. It is constructed using the greedy heuristic outlined in Algorithm 2. The generation heuristic does not consider time shifting, instead it only considers insertion of the original trip from each metatrip. Therefore, when writing *earliest metatrip* in Algorithm 2 we refer to the metatrip whose original trip is the earliest. The heuristic constructs vehicle routes one at a time and attempts to create routes where little time is wasted in between trips. Lines 2–12 deal with the construction of a single route for a vehicle. Lines 2–4 select the first trip on the route and the depot which should provide the vehicle for the route. Lines 5–12 add trips to the partial route. The selection of which trip to add is based on the terminal where the partial route is ending at the moment. The algorithm adds the first trip that leaves that terminal or closes the route if the route cannot be extended with a trip starting in the current terminal.

C.5 Data

The data set that has been developed for the SVSPSP during the preparation of this paper has been described in further detail in a technical report by Petersen et al. [92], and in this section we will give a brief description of the background and the resulting data set. The data set can be obtained online[2].

The local train network in the Greater Copenhagen area roughly has the

[2] `http://www.transport.dtu.dk/SVSPSP/`

form of a fan or the fingers of a hand, as shown in Figure C.5. A network of express bus lines complements the train lines across and in parallel, as can be seen in Figure C.6. The data set that has been developed for the SVSPSP is based on this structure, where the radial train lines are operated on a fixed timetable, and the timetables for the bus lines (of which most are circular) are adjusted according to this.

Figure C.5: The local train network of Copenhagen.

A data set for the SVSPSP consists of several parts: 1) a *distance matrix*, containing all distances between depots and line end-points, 2) *fixed timetables* of all fixed-schedule train connections, 3) *number of transferring passengers* for each transfer opportunity, 4) *an initial schedule* used to determine the available set of trips, 5) *costs* of different activities, and other parameters such as turnaround times, passenger transfer times, etc.

Among these elements the distances and fixed timetables are generally relatively easy to obtain. Furthermore the initial schedule, in the form of the current bus schedule, is required to provide information regarding

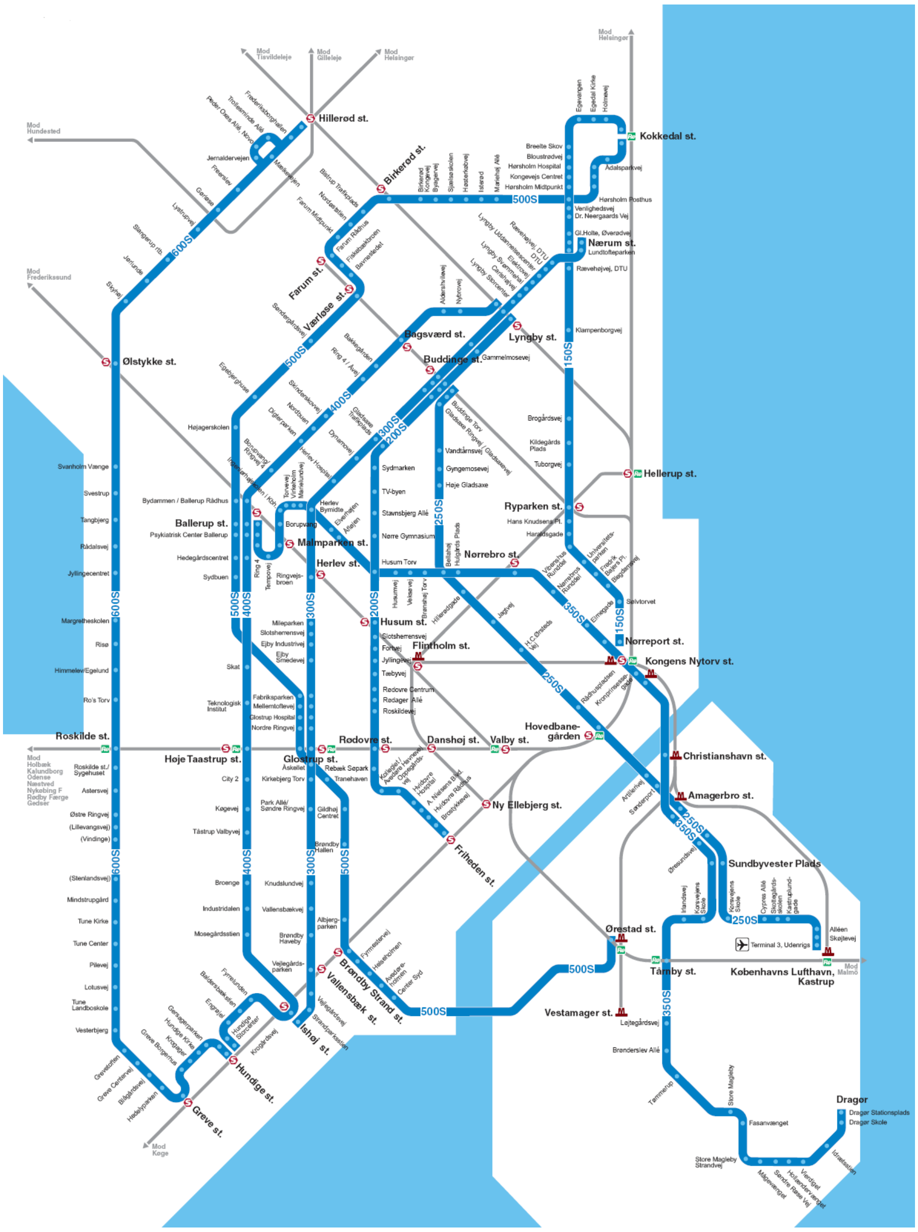


Figure C.6: The S-bus network; trains are shown as thin lines, compare Figure C.5.

frequencies and service level, which will be maintained by the new solution. Given a suitable generation strategy, the set of potential trips can be generated based on these time tables. The current schedule can also be used to generate an initial feasible (VSP) solution for the heuristics.

The problem objectives of operating cost and passenger waiting time have been combined by expressing both in monetary units. The various costs required for calculating the total cost of a solution have been estimated for the data set, in particular the cost of passenger waiting time has been estimated based on the value of travel time recommended by the Danish Ministry of Transport.

What then remains to be estimated is the number of passengers and their transfer patterns. This transfer information will allow us to calculate the number of (dis)embarking passengers using each available transfer opportunity, for any arrival or departure of a bus at a station.

For this project these data have been obtained by a two-stage process: First we estimated the number of (dis)embarking passengers, as a function of the station, bus line and time of day, and then we estimated the percentage of (dis)embarking passengers that would perform each possible transfer.

The number of (dis)embarking passengers at each station is calculated as $f_t \cdot f_l \cdot f_s \cdot n$ where f_t is a time factor, f_l is a line factor, f_s is a station factor, and n is a random number evenly distributed in the interval $[32, 48]$. The values of n is chosen to roughly reflect the capacity of a vehicle, and the introduction of randomness increases the variation of data, to make them more realistic.

The distribution of transferring passengers between available connections has been estimated based on knowledge of the network, and considering the direction of trains (towards the town centre or away from it). Again a random element has been added to provide a better spread of the obtained values. Connections have been specified either for a particular train line or as e.g. "the first departure going into town". For modelling purposes this could be obtained by adding artificial train lines.

Metatrips are created from trips in the original timetable. Let T_i be the departure time of a trip in the original timetable, belonging to a particular d-line L. We create an interval $[T_i^s, T_i^e]$ around T_i and distribute κ trips in this interval to form a metatrip. Assume that κ is an odd number. We express the start and end of the interval as follows $T_i^s = T_i - \tau_i^-$ and $T_i^e = T_i + \tau_i^+$. The symbols τ_i^- and τ_i^+ are expressed in terms of the departure times T_{i-1} and T_{i+1} of the previous and next trips, respectively, on L as follows: $\tau_i^- = \lfloor \frac{T_i - T_{i-1} - 1}{2} \rfloor, \tau_i^+ = \lfloor \frac{T_{i+1} - T_i}{2} \rfloor$. This construction ensures that the intervals around the trips on each d-line are disjoint. The set of departure times constructed are

$$\left\{ T_i - \frac{2j}{\kappa}\tau_i^- : j = 1, \ldots, \left\lfloor \frac{\kappa}{2} \right\rfloor \right\} \cup \{T_i\} \cup \left\{ T_i + \frac{2j}{\kappa}\tau_i^+ : j = 1, \ldots, \left\lfloor \frac{\kappa}{2} \right\rfloor \right\}$$

with the time expressions rounded to the nearest integer to ensure that departures occur at integer valued times. If the trips in the original timetable are close then we may end up with fewer than κ departure times because some departures get mapped to the same integer due to the round-off. In that case we only create as many trips as we have departure times for. In our test we used $\kappa = 5$. Figure C.7 shows an example of how the trips of a metatrip are distributed.

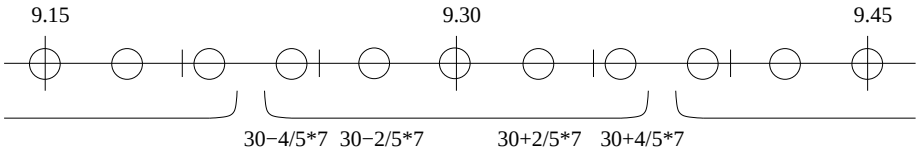


Figure C.7: Example of the distribution of trips in metatrips.

The only incompatibilities used in this project, have been found by multiplying the current interval between two trips by a factor to determine lower and upper bounds allowed for the same interval. This factor has been set to 0.5 for the lower bound and 1.5 for the upper bound.

Instances of three different sizes have been considered for this project. These instances have been constructed by considering a meaningful subset of the actual operated bus routes, i.e. a subset that in itself constitutes a realistic problem. This means that the set of routes selected for the smaller subset have characteristics that may differ from the routes added in the larger subsets. Thus the smaller problems consist of the most central lines, and the lines that are added in the larger sets are more rural, and/or have fewer intersections with the train network.

The properties of the three different instances can be summarised as follows:

3 lines. 538 trips. All lines are circular lines with 5–6 intersections with the train network, but only few interconnections between the buses. Many passengers. Subset of

5 lines. 792 trips. All lines are circular lines with 4–6 intersections with the train network, and only few interconnections between the buses. Some lines are passenger intensive. Subset of

8 lines. 1400 trips. Combination of circular and radial lines. The radial lines only have 2–3 connections to trains, but more connections to other buses. Most lines are passenger intensive.

C.6 Computational experiments

To evaluate the quality and usefulness of the algorithm, we have performed a series of tests to examine its behaviour with different instance sizes and settings, which will be presented in this section. The tests have been performed on an Intel Pentium 4, 2.8 GHz, with 2GB RAM,

running Windows XP.

The current vehicle schedules used for the data set were not available, so these had to be constructed initially. This has been done by using the implemented LNS as a regular VSP solver, i.e. by not allowing any time shifts. The generated solutions have been used as initial solutions when solving the SVSPSP, and also as reference solutions representing current practice, when evaluating the quality of the obtained final solutions. As we know that the actual current schedules are not created with dedicated software, this should produce reference solutions that are not worse than the currently used solution. For each instance a running time of 24 hours was allowed for the construction of the reference solution.

Table C.1 shows the results from running the implemented LNS algorithm on instances of different sizes with different running times. For

3 lines

	total cost	veh.	empty	time	shifts	avg. shift	reg.
1h	2.9%	0.0%	−14.2%	16.5%	74.2%	2.19	39.7%
6h	3.1%	0.0%	−13.0%	17.4%	73.4%	2.22	43.2%
24h	3.3%	0.0%	−8.9%	18.1%	73.8%	2.11	48.1%

5 lines

	total cost	veh.	empty	time	shifts	avg. shift	reg.
1h	2.8%	0.0%	−10.1%	19.8%	77.0%	2.58	39.8%
6h	3.1%	0.0%	−9.2%	21.8%	79.3%	2.68	43.4%
24h	3.2%	0.0%	−7.8%	22.5%	78.2%	2.61	40.5%

8 lines

	total cost	veh.	empty	time	shifts	avg. shift	reg.
1h	1.1%	0.0%	−7.8%	9.5%	64.2%	1.88	30.4%
6h	1.6%	0.0%	−6.4%	13.3%	76.6%	2.38	31.4%
24h	2.0%	0.0%	−7.1%	16.4%	76.4%	2.39	36.0%

Table C.1: Solution improvements for different problem sizes.

each run we report the cost reduction compared to the initial solution, the number of vehicles used, the reduction of empty mileage costs (i.e. a negative value indicates that the empty cost has increased), the reduction of total passenger waiting time, the percentage of trips that have been time shifted, the average amount of time that each trip is shifted, and the percentage of trips that are *regular*. A *regular*/memorable trip is a trip for which the gap to the preceding trip on the same line is a multiple of 5. This makes the schedule easier to remember, and is thus

an advantage to the passengers. For the current schedule the percentage of regular trips is around 72% for the largest instance, and 83–84% for the others. However, memorability has not been an objective of the implemented algorithm.

The table shows that good results can be obtained, and that a considerable reduction of passenger waiting time is possible. The reduced waiting times lead to an increase in the amount of empty travel, however the total operating cost still shows improvement of around 3% for the smaller instances, and 1–2% for the 8 line instances.

Alternative small instances

As stated previously the different tested instances differ not only in size, but also in some characteristics regarding the type of lines that are used. Thus the variation in cost and time reduction obtained for the different instances may well depend just as much on the change in these characteristics as on the actual size of the problems. The tests of Table C.1 have been repeated on two additional small instances that have been created with a mix of lines more similar to those of the largest instance. These instances represent subproblems that would most likely not be considered in real-life, but can hopefully demonstrate the behaviour on smaller instances without being affected by the different characteristics of the problem. Each instance consists of two circular lines (of which one is passenger intensive) and one radial line. The results for these two instances can be found in Table C.2, and indicate that it is difficult to compare the properties of instance just by looking at simple properties of the included lines. The results also indicate that the achievable cost improvement does indeed depend on the choice of lines to include in the problem.

	total cost	veh.	empty	time	shifts	avg. shift	reg.
1h	1.3%	0.0%	−7.2%	12.2%	73.2%	2.0	29.8%
6h	1.6%	0.0%	−7.7%	14.7%	76.4%	2.1	31.4%
1h	2.9%	0.0%	−8.6%	20.4%	79.4%	2.8	39.2%
6h	3.1%	0.0%	−5.6%	21.3%	76.5%	2.8	45.0%

Table C.2: Solution improvements for more “realistic” small instances.

Random variation of the instances

The network structure and the existing time tables are fixed, so in order to produce a series of different data sets/problem instances that still reflect the real world, the only adjustable parameter has been the random element of the spread of the passengers over different available connec-

tions. This has been done for the medium-sized instances (5 lines), using running times of 1 and 6 hours, and the results can be found in Table C.3.

	total cost	veh.	empty	time	shifts	avg. shift	reg.
1h	2.8%	0.0%	−10.5%	19.7%	78.8%	2.7	37.3%
	2.2%	0.0%	−6.4%	15.4%	75.5%	2.5	39.3%
	2.8%	0.0%	−11.8%	20.1%	77.8%	2.7	34.5%
	2.7%	0.0%	−11.6%	19.7%	76.8%	2.7	39.6%
6h	3.2%	0.0%	−6.2%	21.8%	76.4%	2.6	39.9%
	2.6%	0.0%	−4.8%	17.8%	77.1%	2.7	43.1%
	3.1%	0.0%	−9.0%	21.8%	78.3%	2.6	43.1%
	3.2%	0.0%	−5.4%	21.8%	76.4%	2.5	39.5%

Table C.3: Solution results with modified transfer distributions.

These results show that the actual distribution of the passengers to some extent influences the size of the reductions that can be obtained, but also that the cost improvements are consistently around 2.6% for the shorter running times, and around 3% for the 6 hour running times.

C.7 Conclusion

We have introduced a new problem that integrates the timetabling and vehicle scheduling phases in public transportation planning. It does so by simultaneously considering resource costs and passenger waiting time at transfers. The problem has been defined formally and a metaheuristic based on the LNS principle has been designed and tested. The metaheuristic has been tested on a data set based on a subset of the buses serving the Greater Copenhagen area. The results obtained are encouraging: for the full data set we have observed that a 16% reduction of passenger transfer waiting times are possible. This reduction was possible without using more vehicles to provide the service, but an increase in the amount of deadheading was necessary. We consider the increase in deadheading negligible compared to the total cost involved in operating a public transport system and when considering the increased passenger service obtained.

A topic for future research is how to make the timetables produced by the heuristic easier for the passengers to memorise. This could be achieved either by adding a term penalising solutions with low memorability to the objective function or ensuring that blocks of subsequent departures have fixed headway.

Appendix D

Additional figures

D.1 15 order instances

These figures show the optimal DTSPMS solution, together with the optimal TSP solution for each tour.

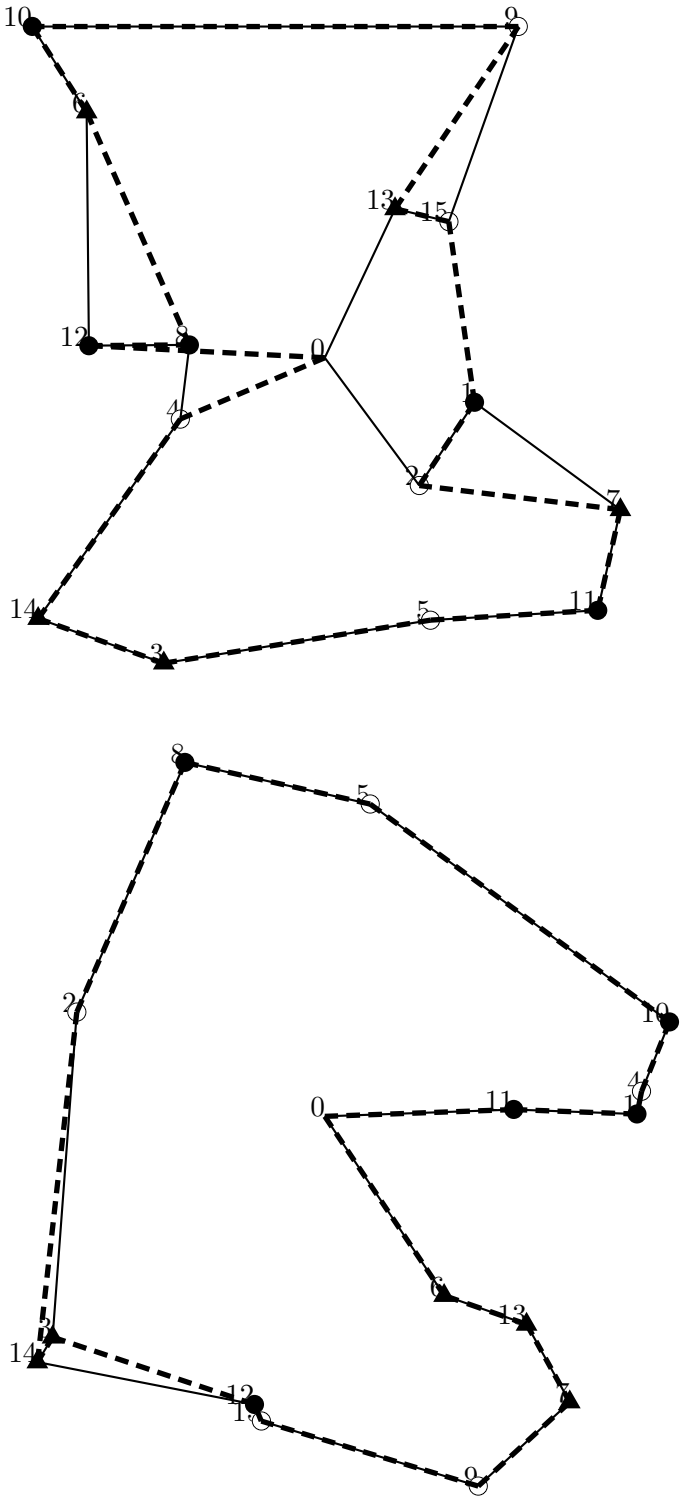


Figure D.1: Instance 00: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (367, 340, 398, 343).

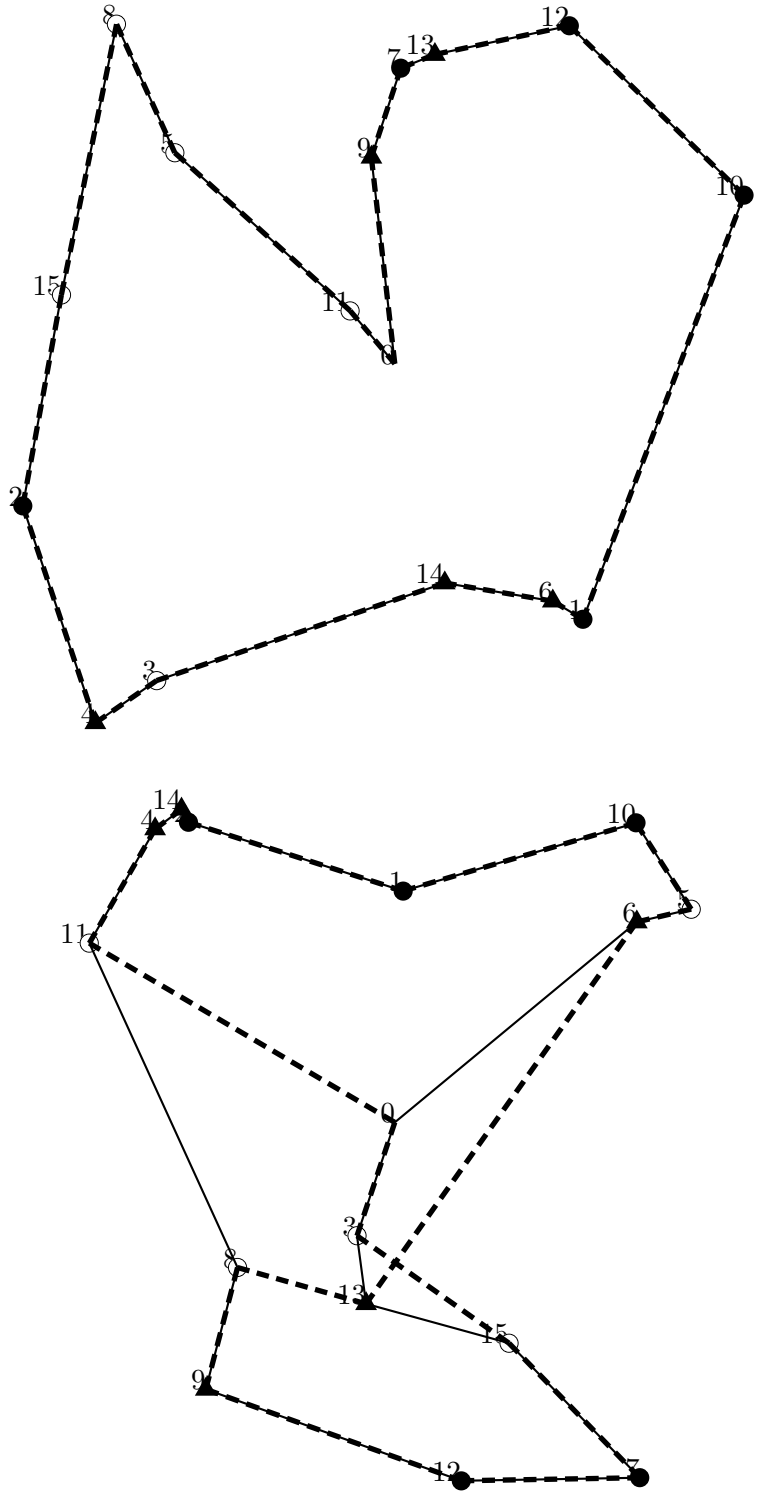


Figure D.2: Instance 01: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (378, 342, 378, 376).

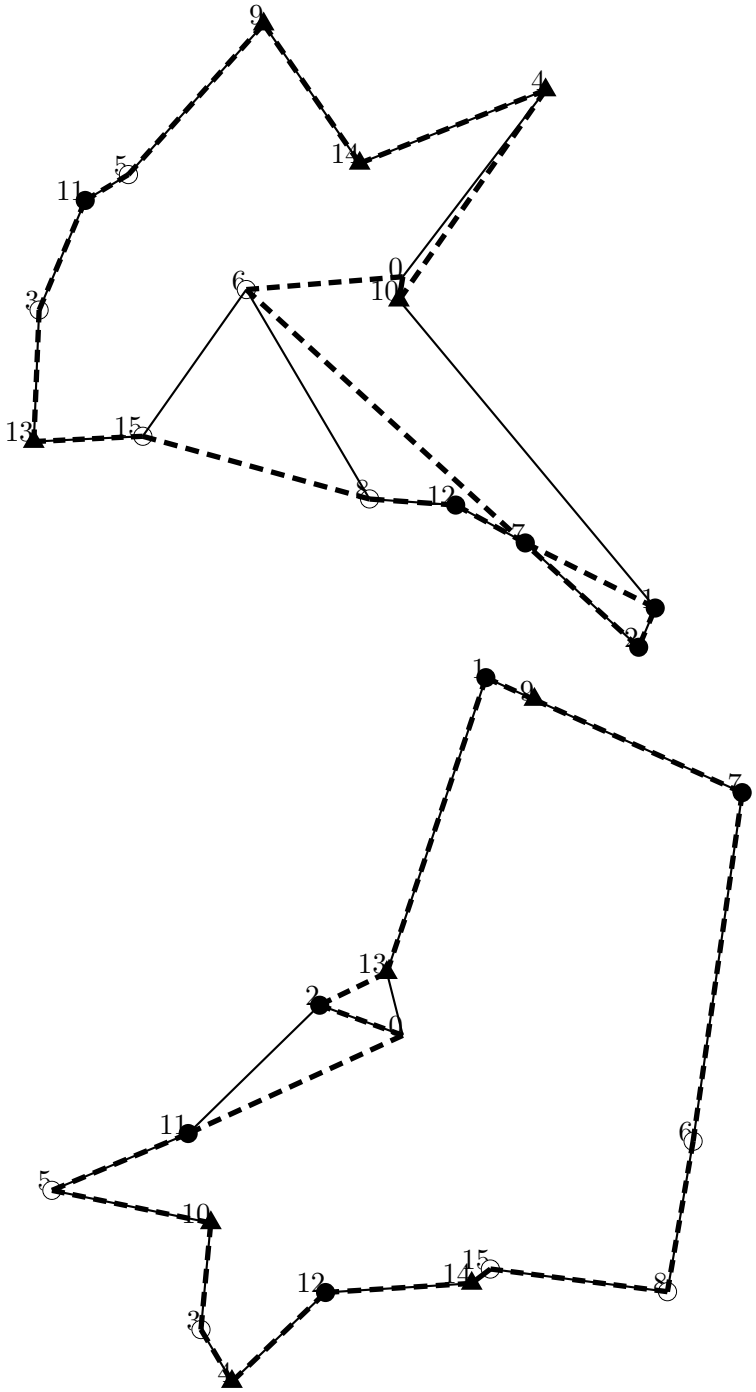


Figure D.3: Instance 02: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (319, 316, 334, 324).

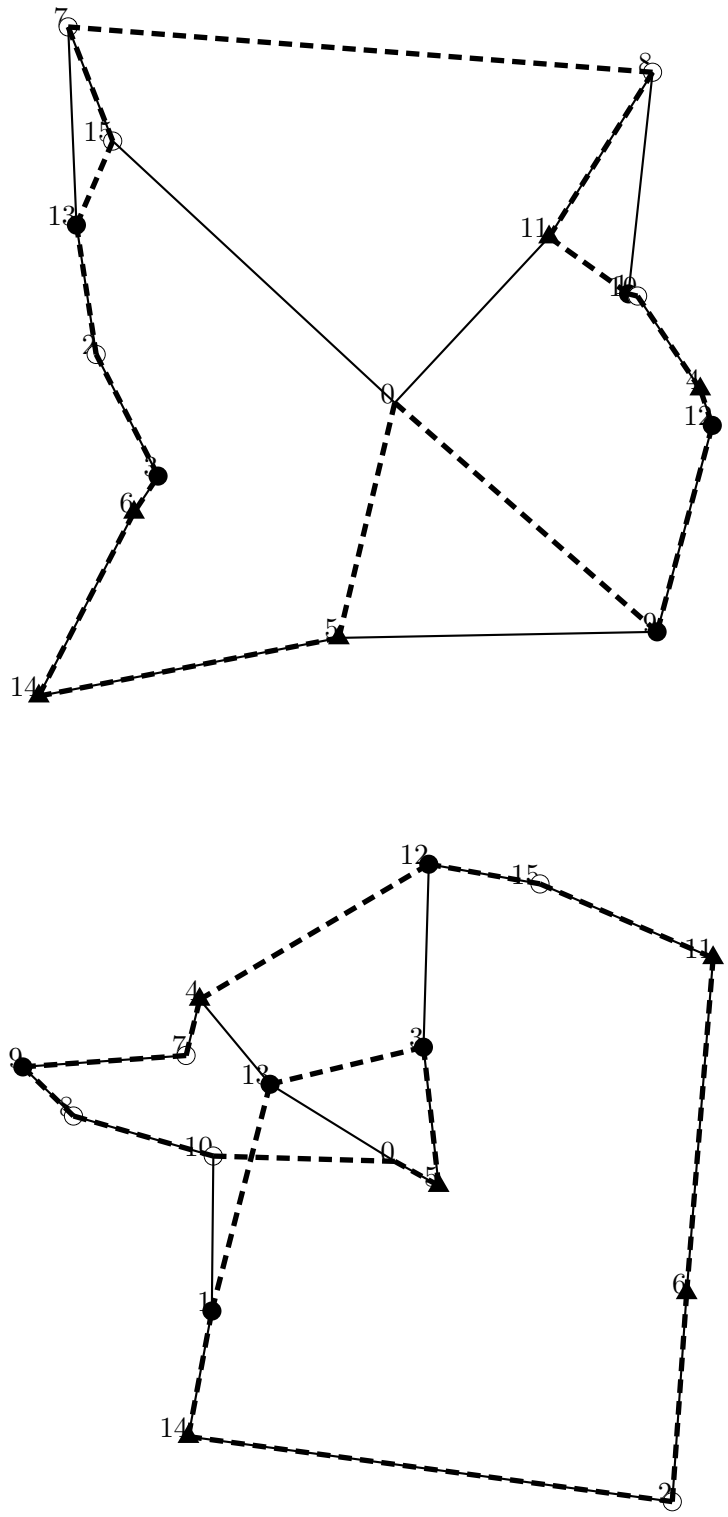


Figure D.4: Instance 03: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (377, 356, 380, 388).

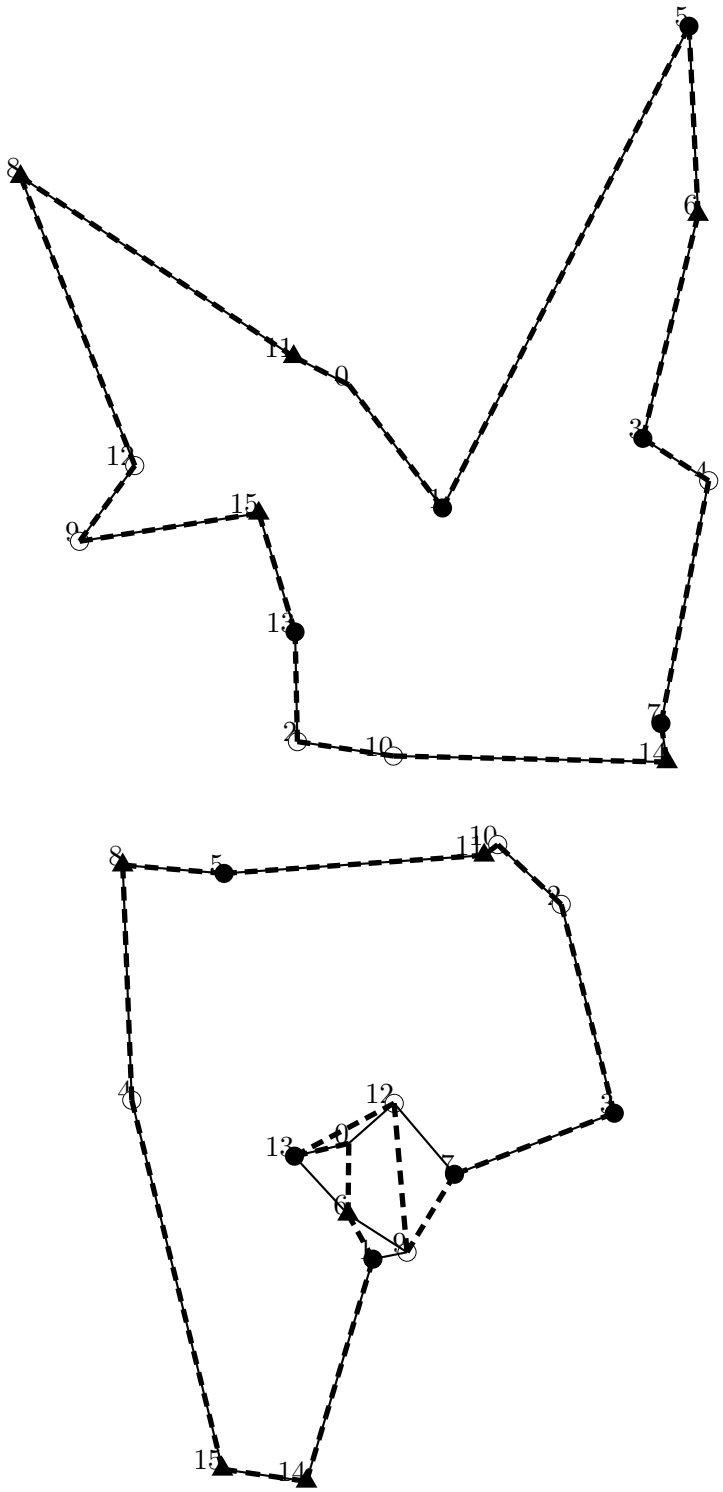


Figure D.5: Instance 04: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (403, 286, 403, 305).

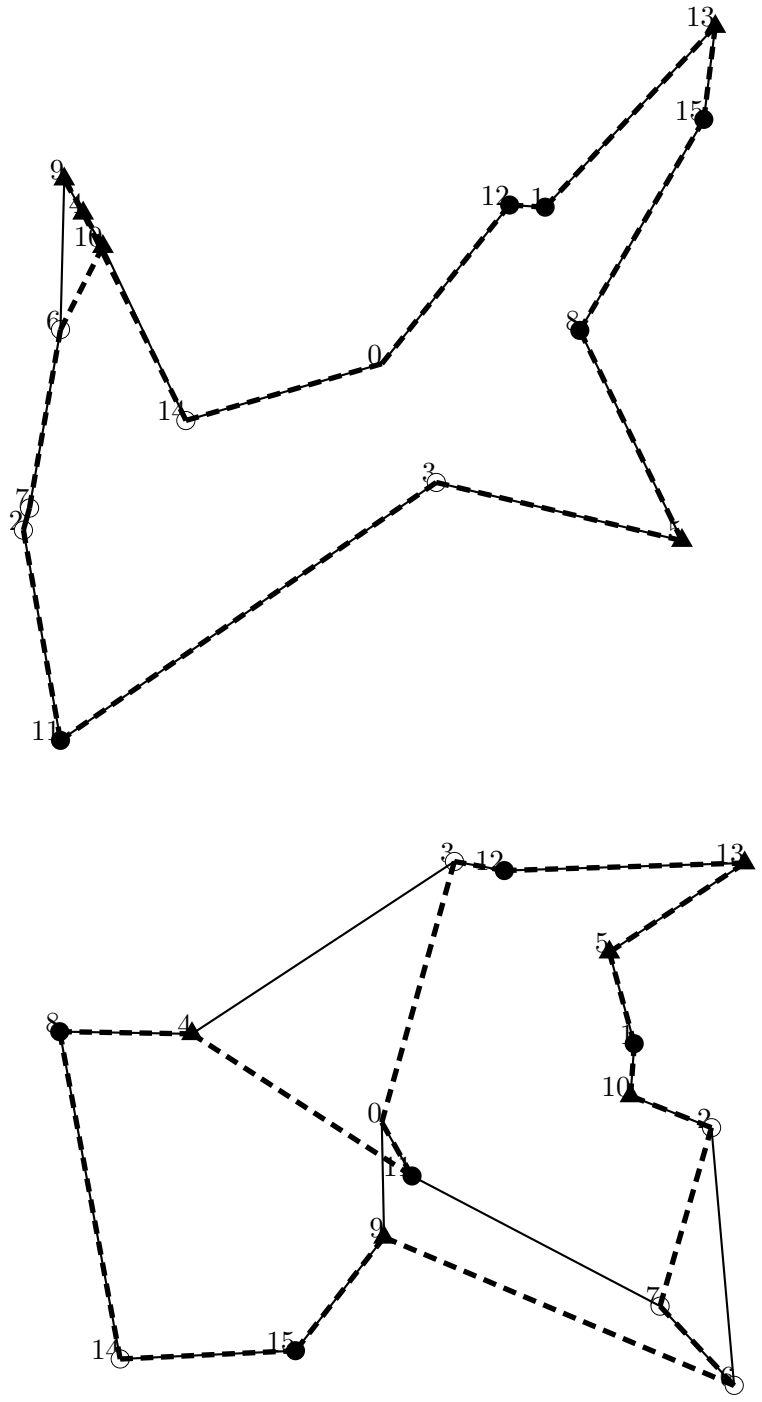


Figure D.6: Instance 05: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (371, 345, 373, 364).

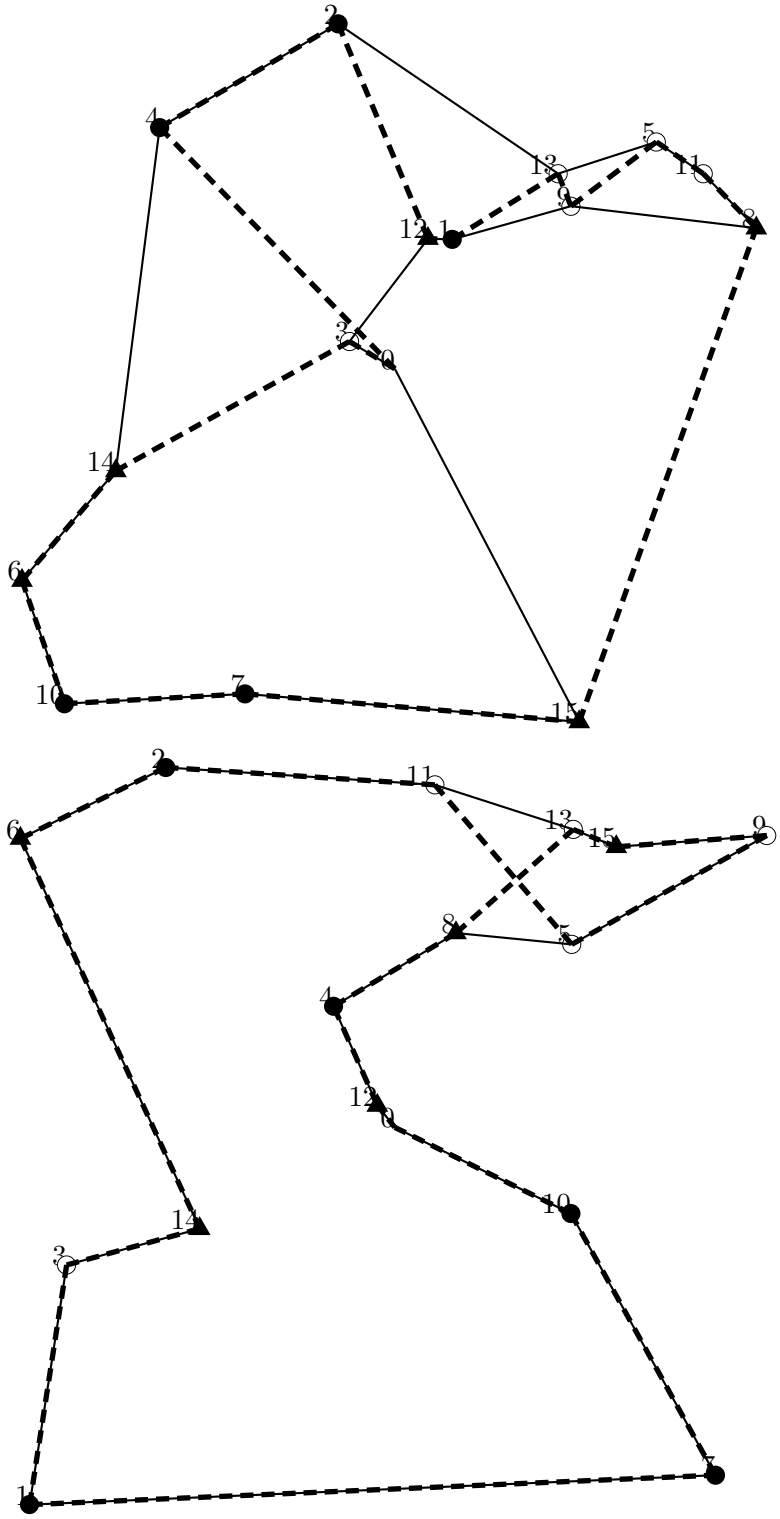


Figure D.7: Instance 06: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (364, 447, 374, 462).

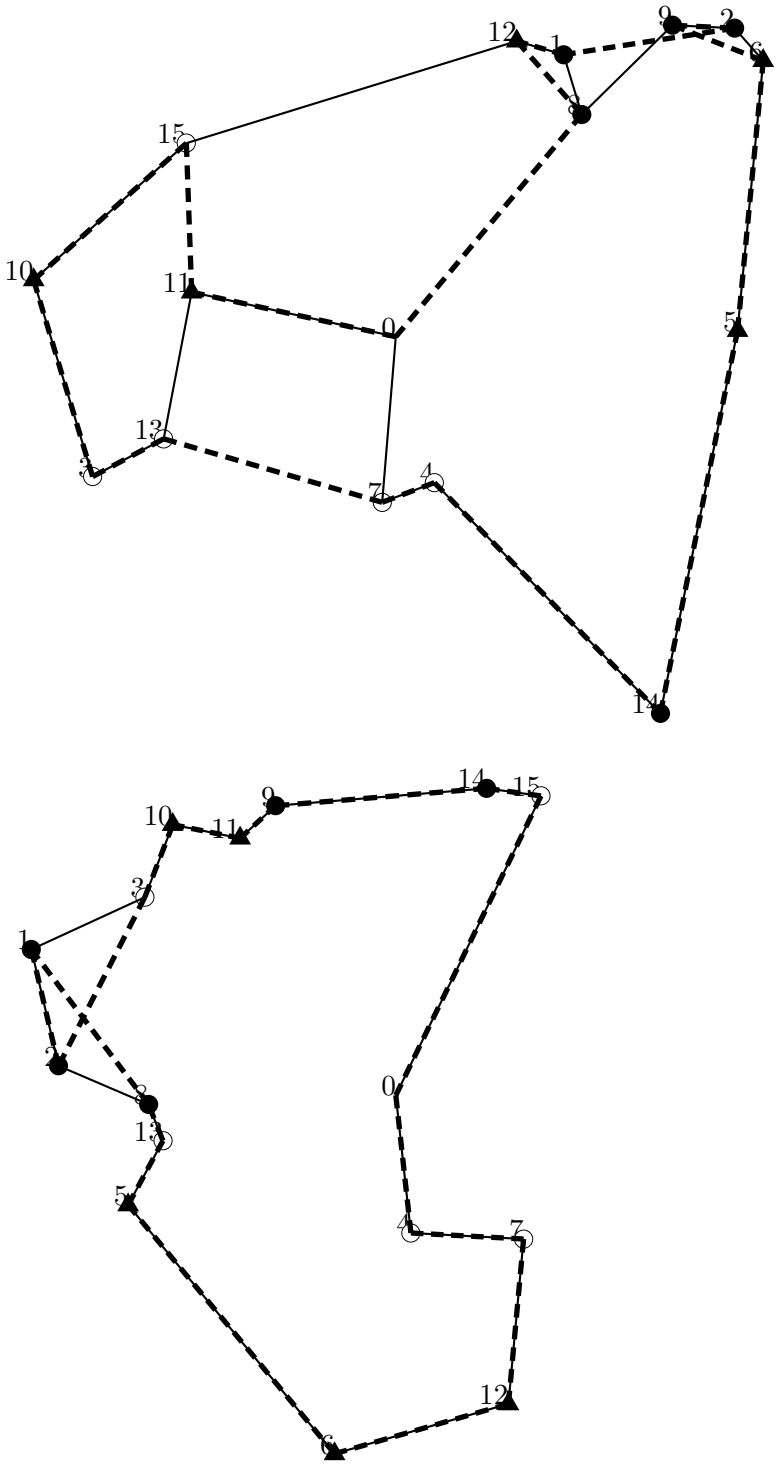


Figure D.8: Instance 07: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (364, 287, 382, 308).

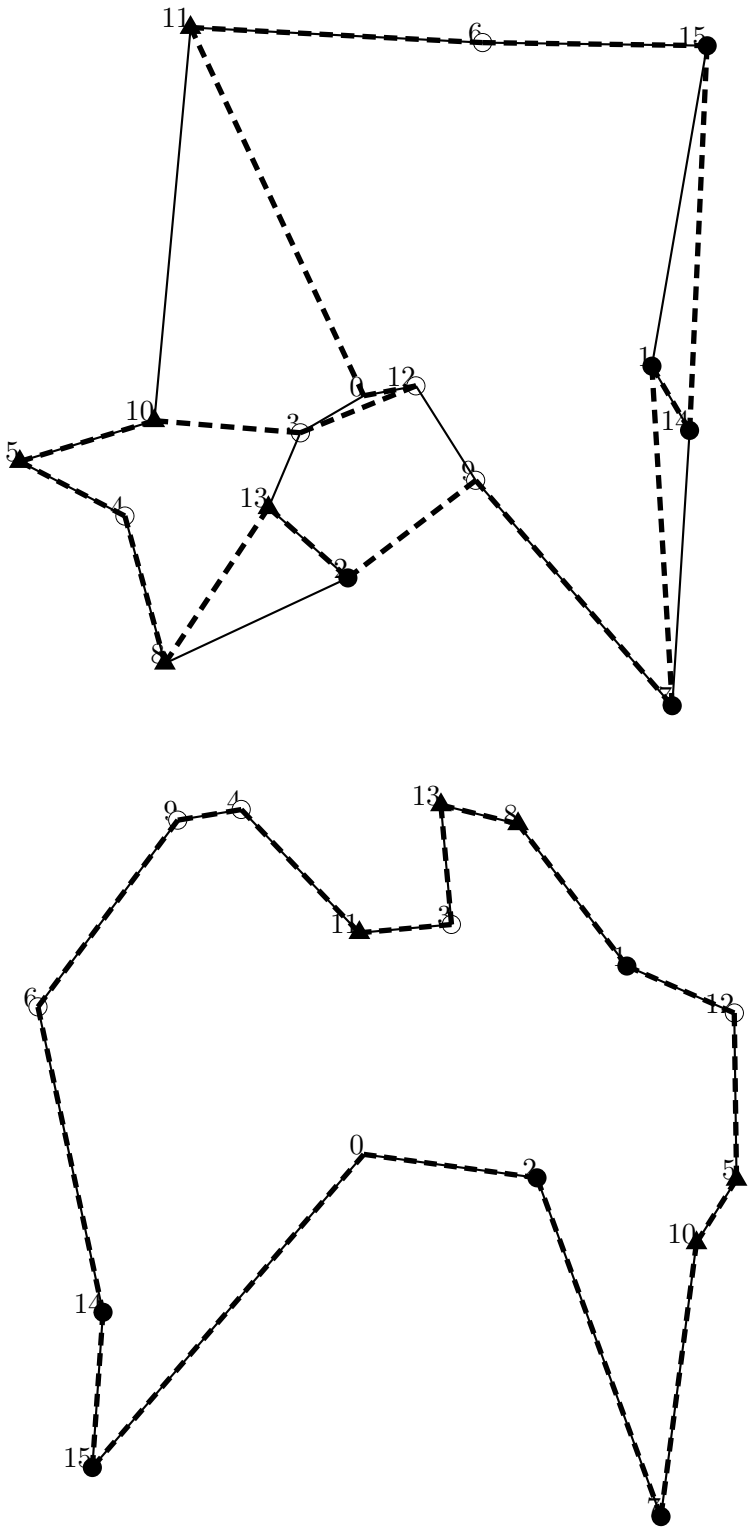


Figure D.9: Instance 08: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (390, 399, 427, 399).

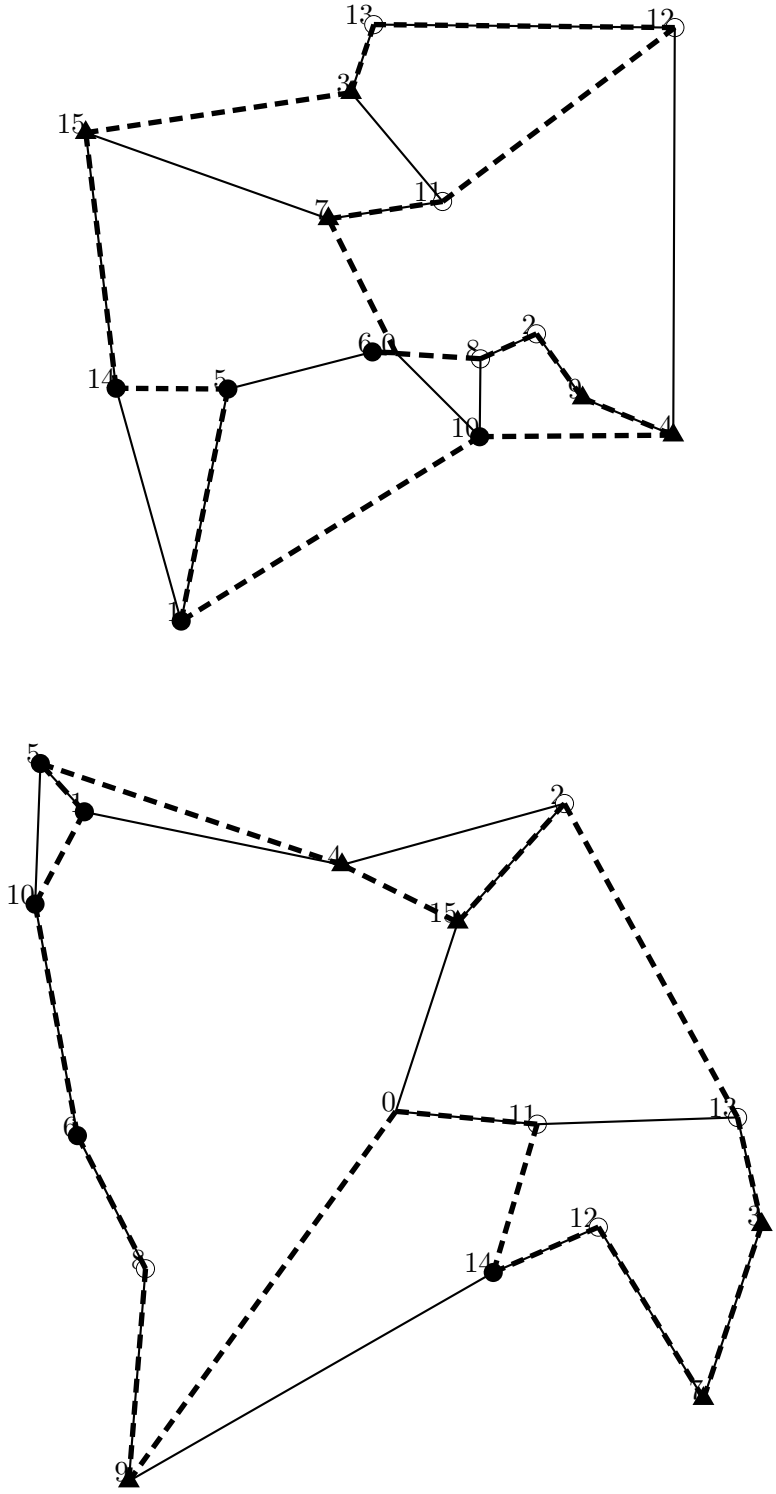


Figure D.10: Instance 09: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (349, 402, 359, 409).

D.2 33 order instances

These figures show the best know solution, together with the optimal TSP solutions.

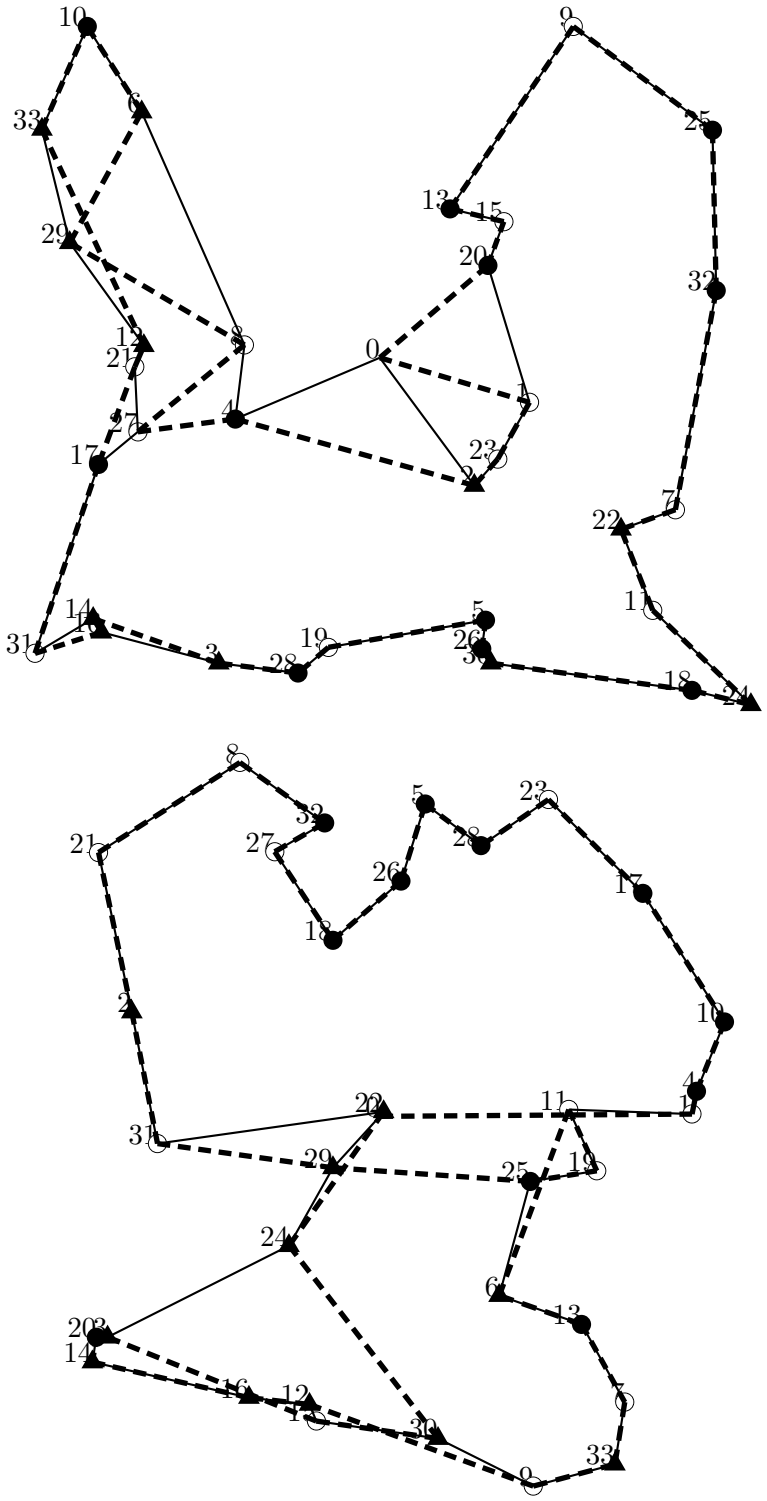


Figure D.11: Instance 00: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (482, 429, 528, 535).

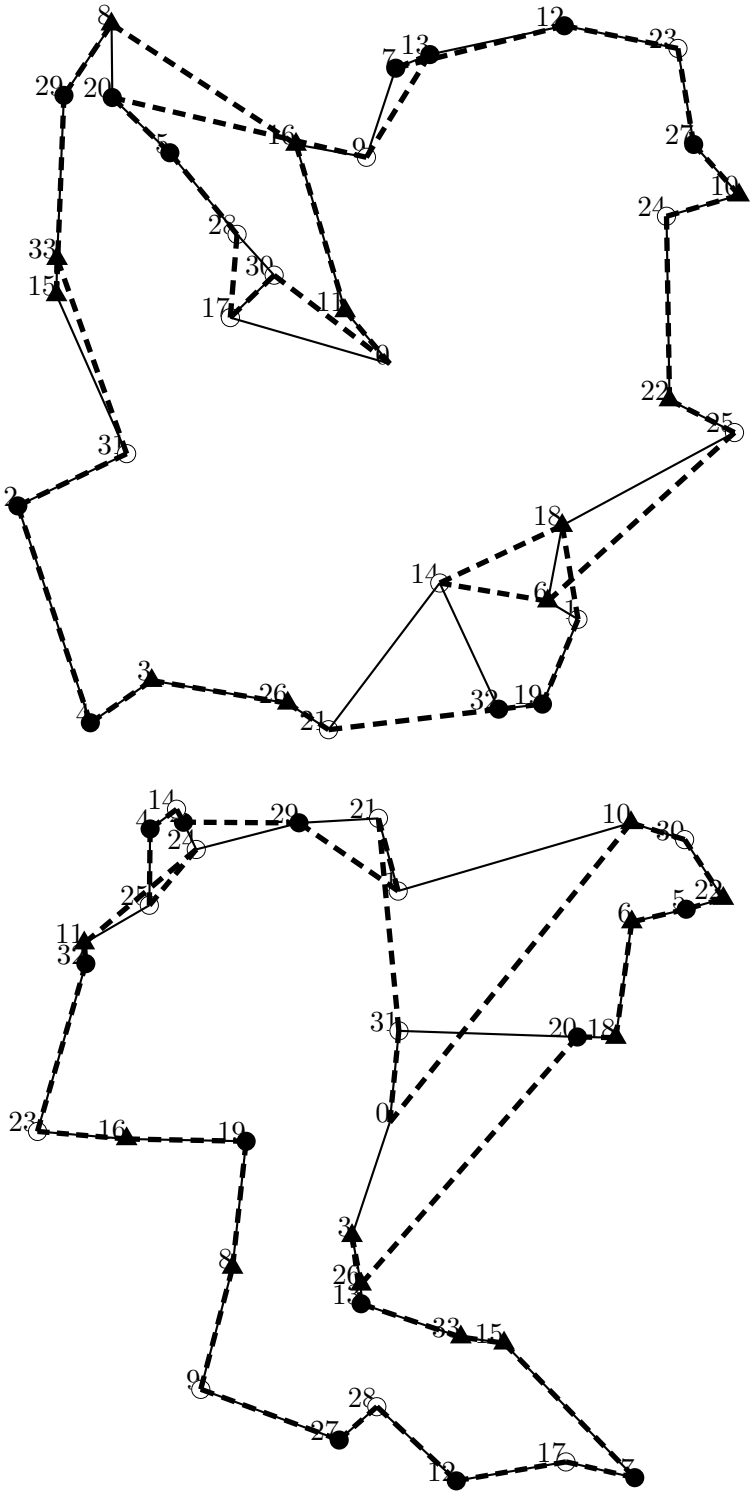


Figure D.12: Instance 01: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (471, 404, 551, 481).

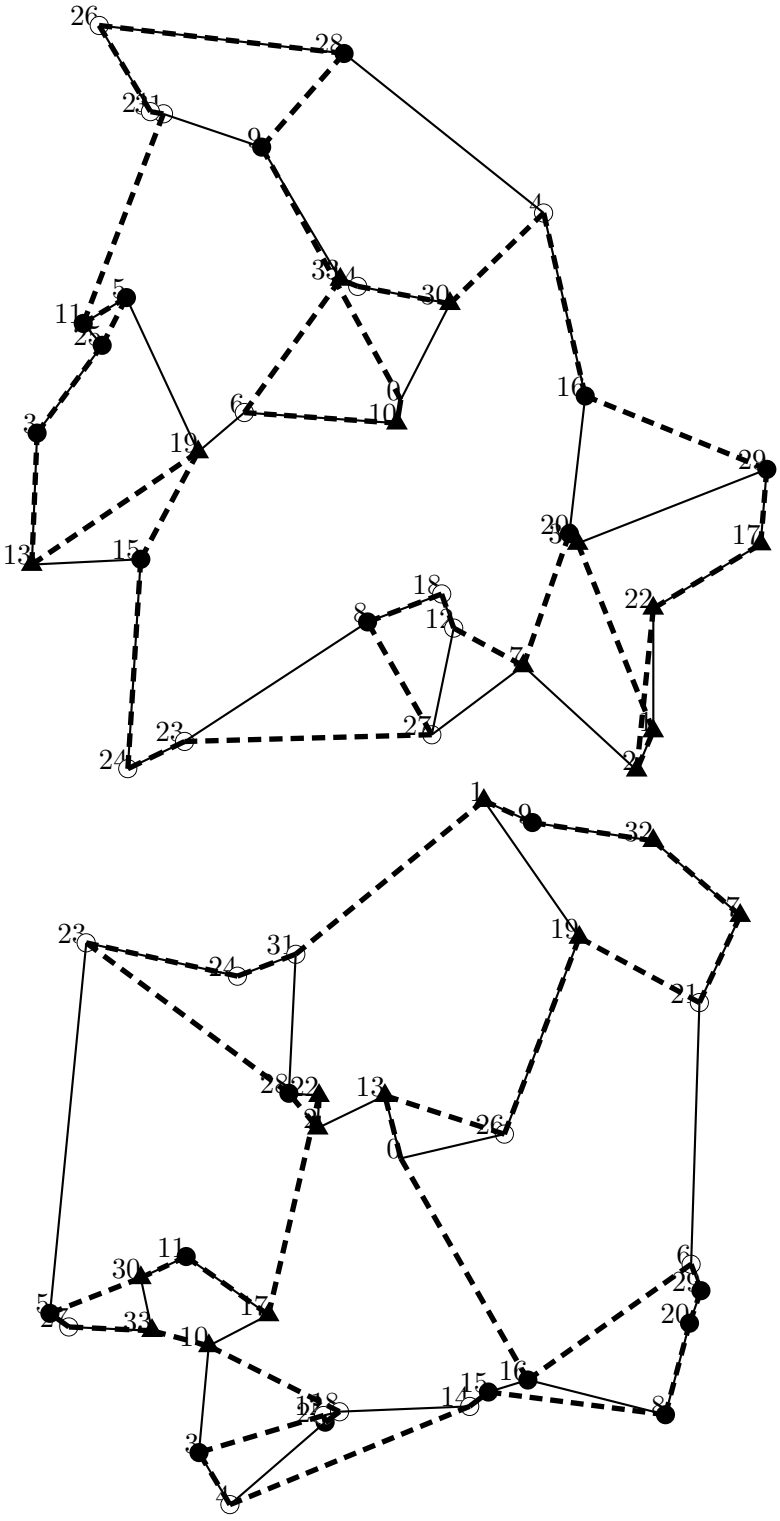


Figure D.13: Instance 02: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (504, 431, 563, 502).

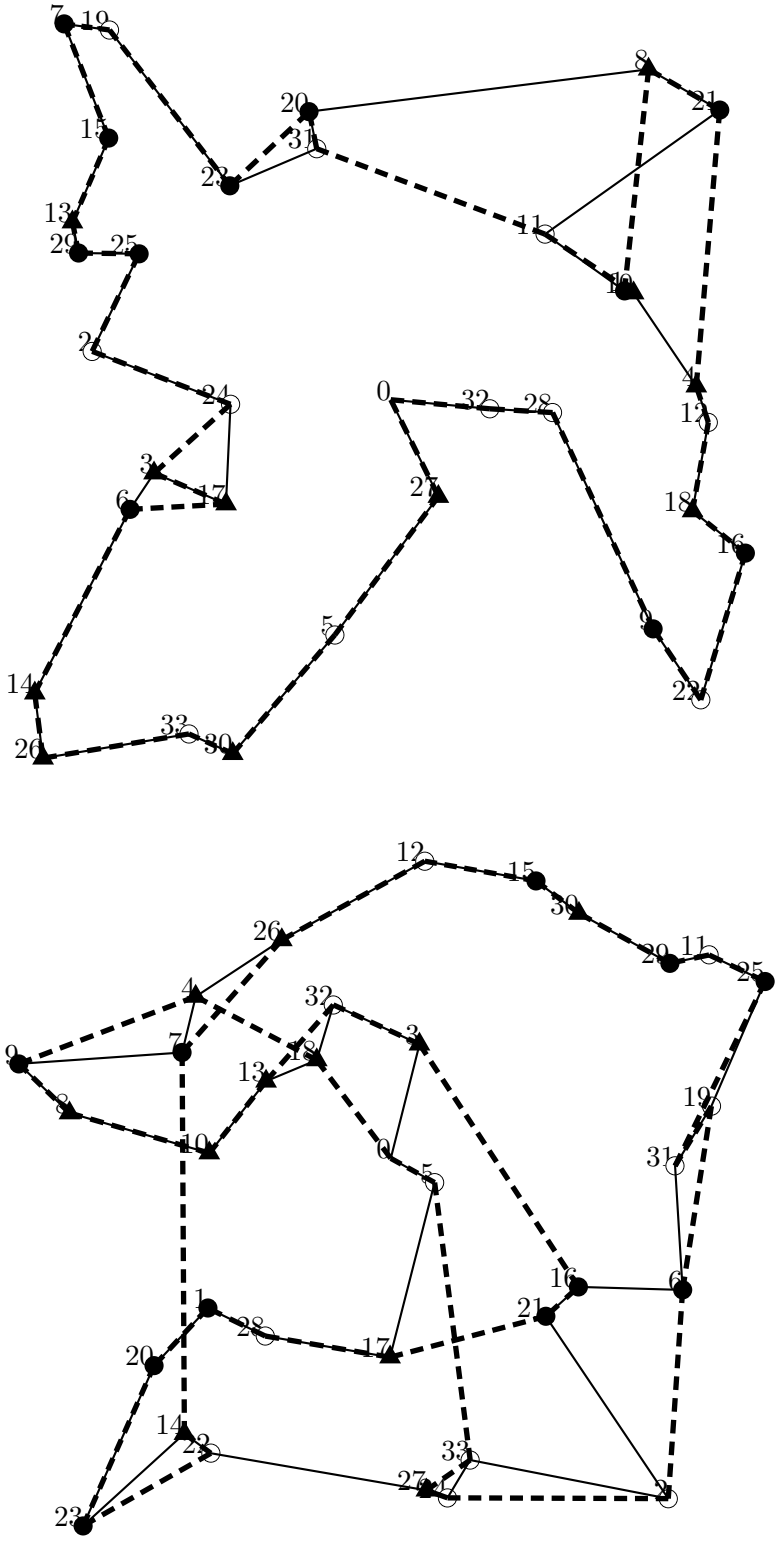


Figure D.14: Instance 03: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (494, 467, 515, 585).

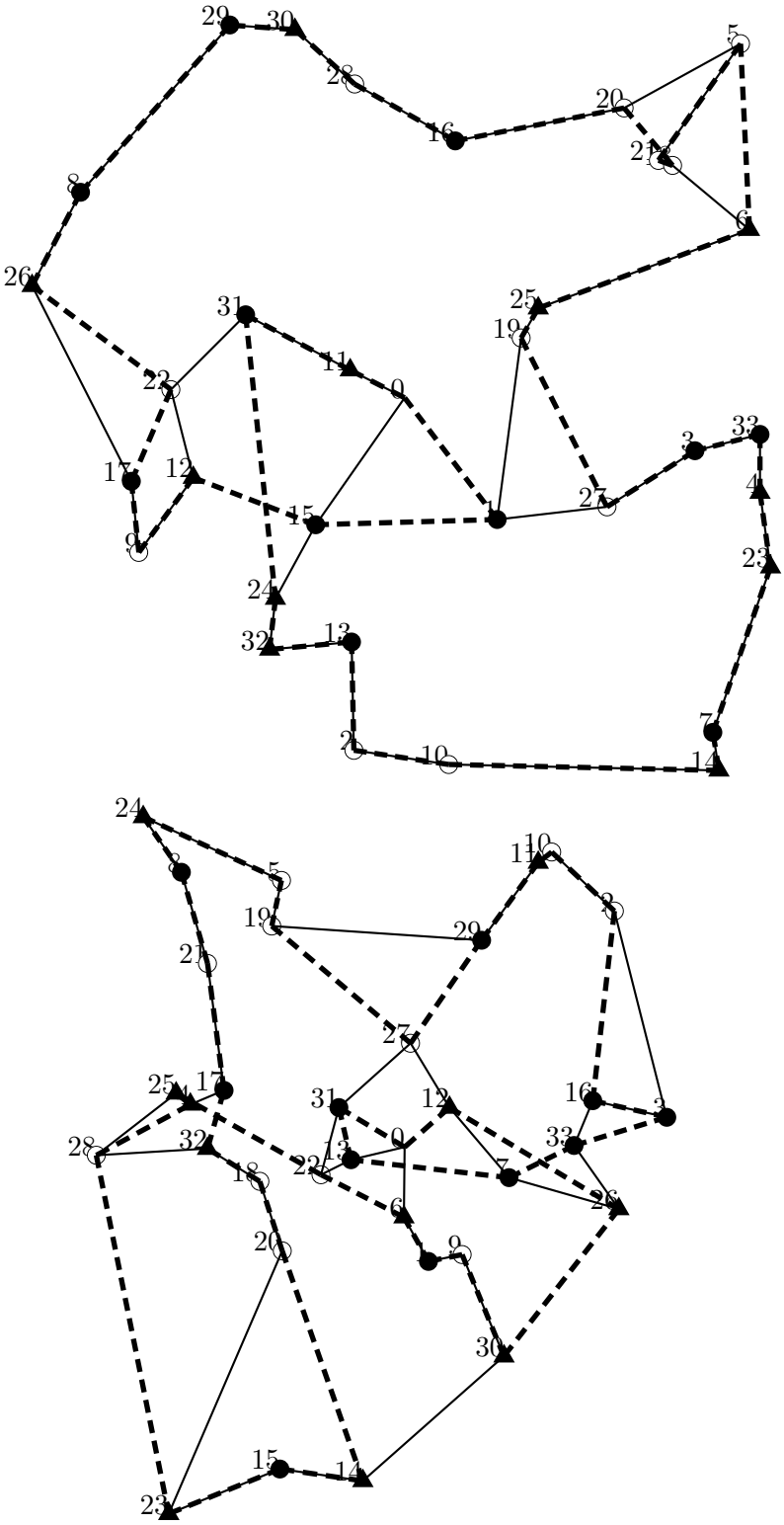


Figure D.15: Instance 04: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (511, 426, 550, 502).

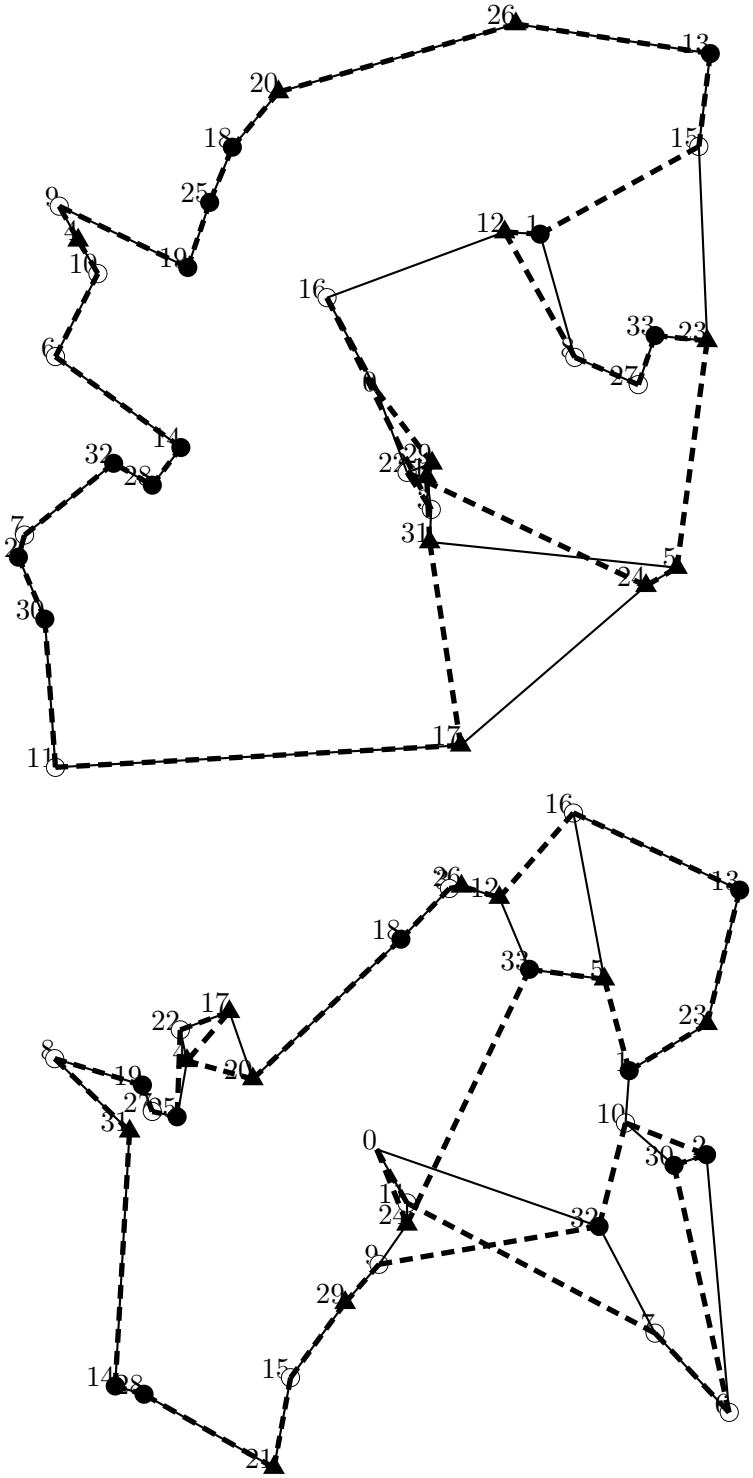


Figure D.16: Instance 05: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (479, 421, 515, 493).

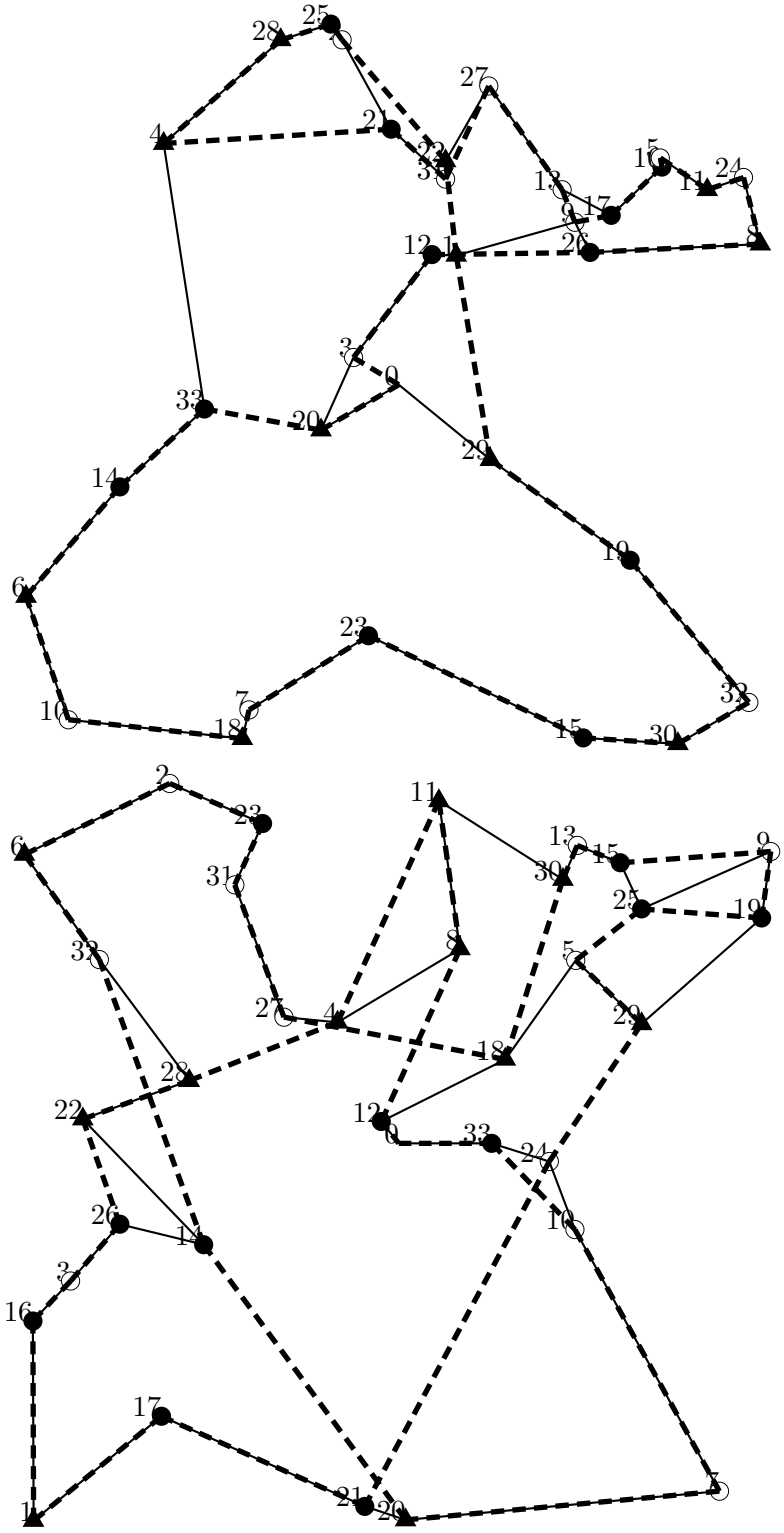


Figure D.17: Instance 06: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (457, 541, 512, 598).

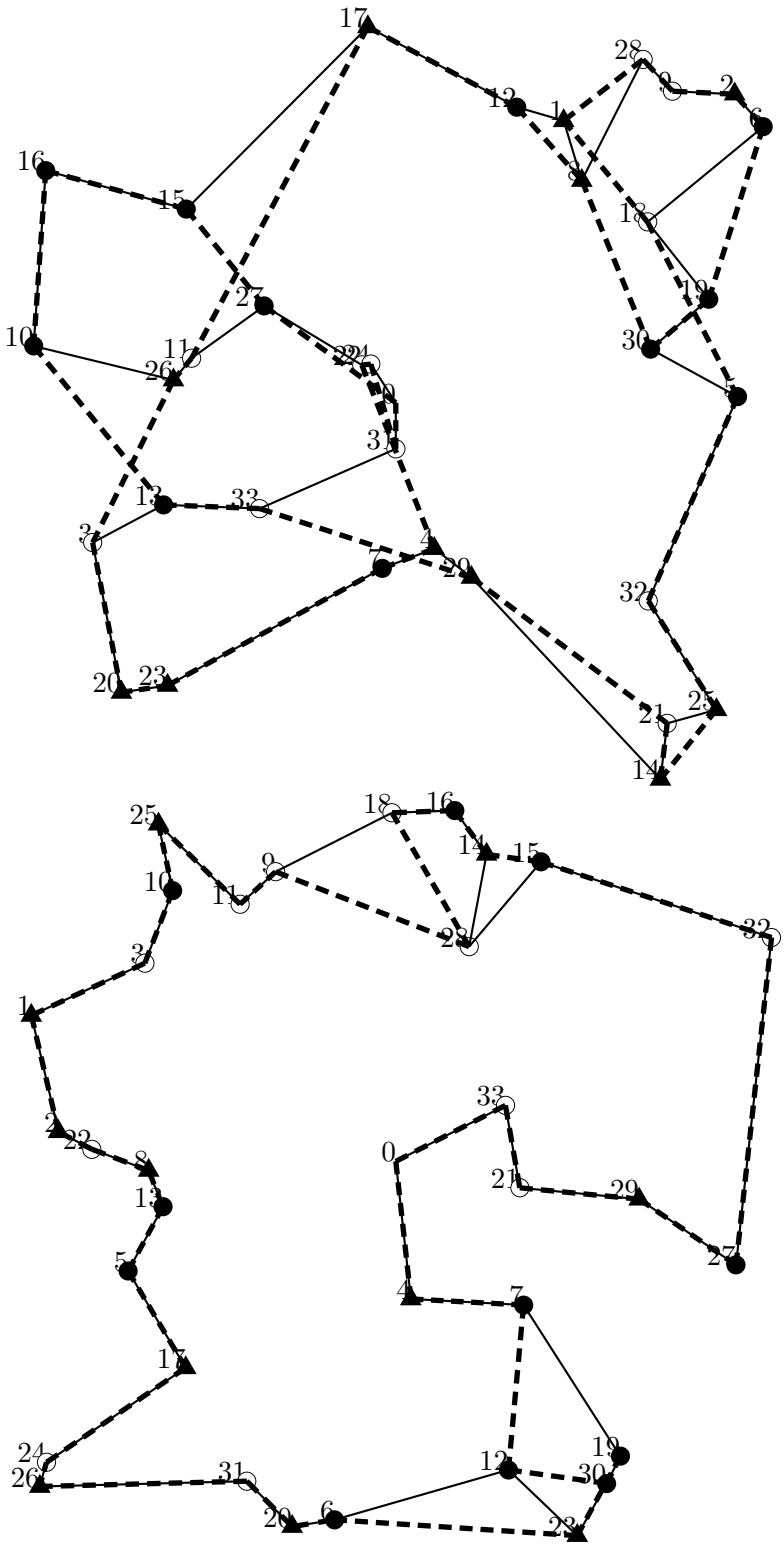


Figure D.18: Instance 07: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (481, 482, 603, 502).

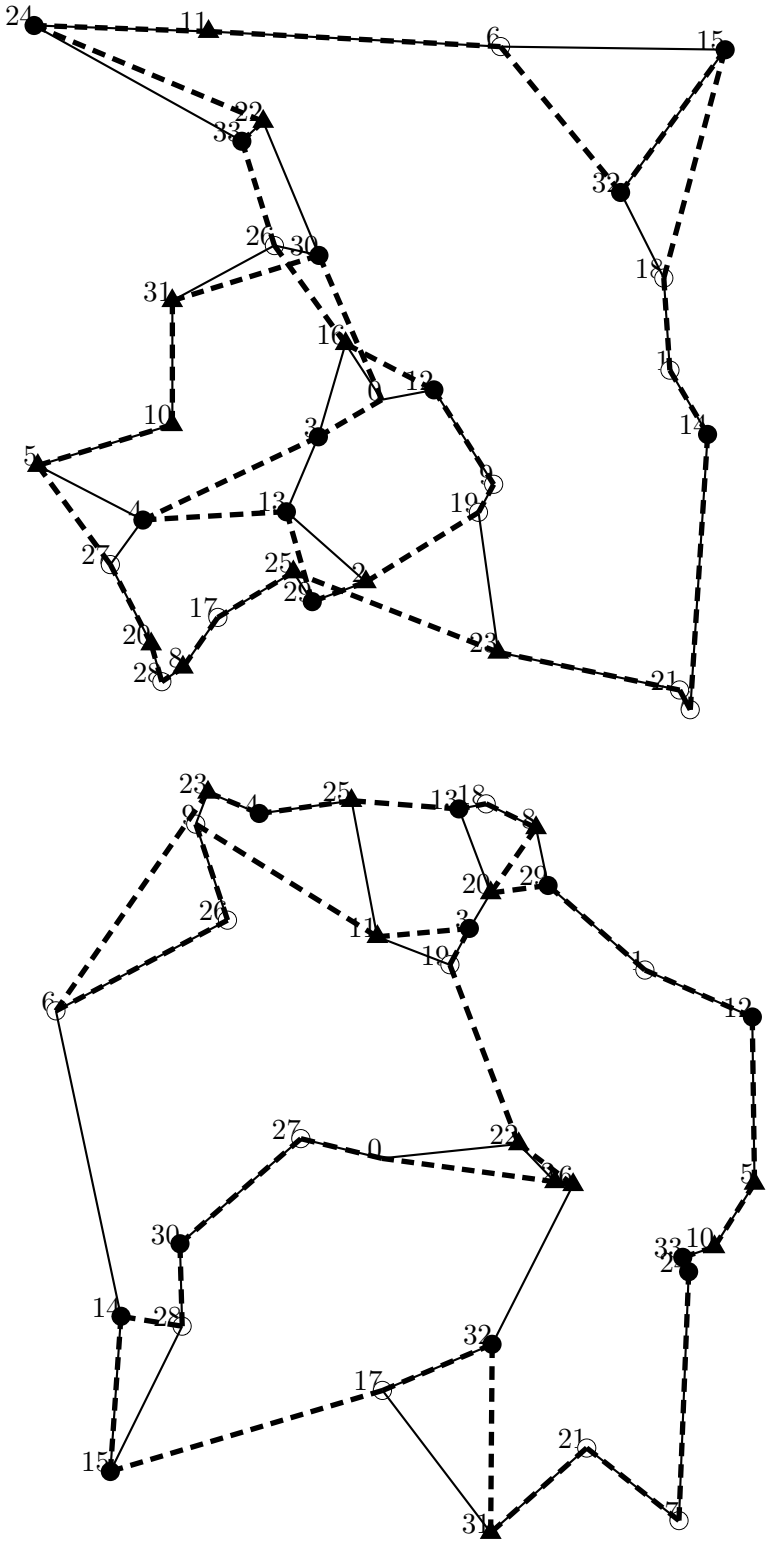


Figure D.19: Instance 08: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (492, 486, 580, 529).

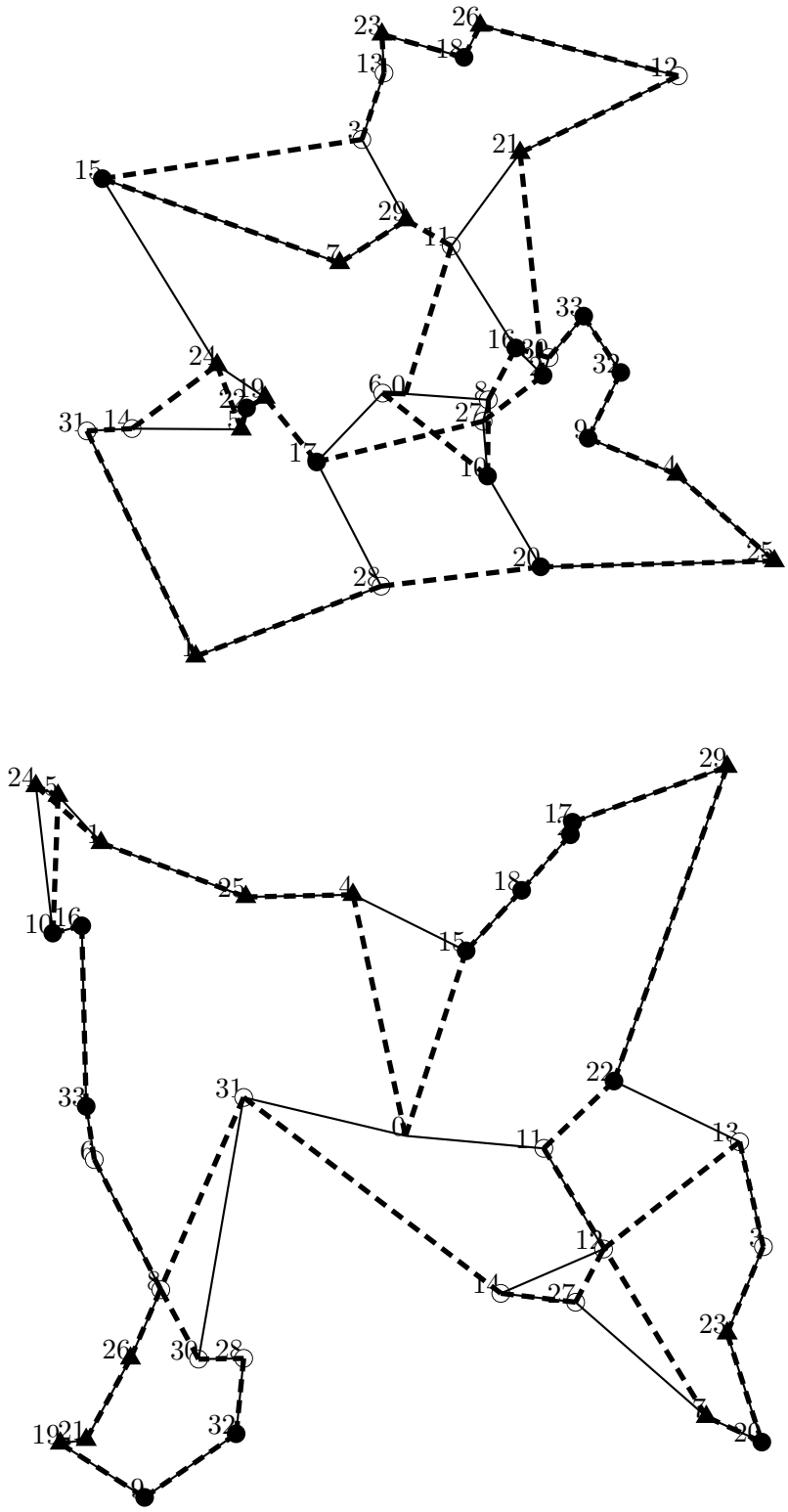


Figure D.20: Instance 09: dashed, fat line shows DTSPMS solution; full, thin line shows optimal TSP solution; objective values for (TSP-P, TSP-D, DTSPMS-P, DTSPMS-D) are (464, 512, 517, 574).

Appendix E

Abbreviations

This appendix contains a list of a number of abbreviations that have been used throughout the thesis.

The abbreviations TSP (Travelling Salesman Problem) and VRP (Vehicle Routing Problem) will be assumed known, and used in the list. The list will explain the name of each abbreviation when written in its entirety. It will not go into discussion of the meaning of the name, or the usage of the name in this thesis.

BWTSP Black and White TSP.
CVRP Capacitated VRP.
DTSPMS Double TSP with Multiple Stacks.
ILS Iterated Local Search.
LNS Large Neighbourhood Search.
MDVSP Multi Depot Vehicle Scheduling Problem.
MPVRP Multi Pile VRP.
PDP Pickup and Delivery Problem.
SA Simulated Annealing.
SVSPSP Simultaneous Vehicle Scheduling and Passenger Service Problem.
SVRPPD Single VRP with Pickup and Delivery.
TS Tabu Search.
TSPPD TSP with Pickup and Delivery.
VNS Variable Neighbourhood Search.
VRPPD VRP with Pickup and Delivery.
VSP Vehicle Scheduling Problem.
TW Time Windows.

Bibliography

[1] R. K. Ahuja, Ö. Ergun, J. B. Ergun, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102, 2002.

[2] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems. *Mathematical Programming*, 97(1–2):91–153, 2003.

[3] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, January 2006.

[4] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. An exact algorithm for the traveling salesman problem with deliveries and collections. *Networks*, 42(1):26–41, 2003.

[5] R. Bent and P. van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530, 2004.

[6] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: a classification scheme and survey. *TOP*, 15:1–31, 2007.

[7] Y. M. Bontekoning, C. Macharis, and J. J. Trip. Is a new applied transportation research field emerging? — a review of intermodal rail-truck freight transport literature. *Transportation Research Part A: Policy and Practice*, 38(1):1–34, 2004.

[8] J. H. Bookbinder and A. Désilets. Transfer optimization in a transit network. *Transportation Science*, 26(2):106–118, 1992.

[9] M. Bourgeois, G. Laporte, and F. Semet. Heuristics for the black and white traveling salesman problem. *Computers & Operations Research*, 30(1):75–85, 2003.

[10] A. Caris, C. Macharis, and G. K. Janssens. Planning problems in intermodal freight transport: Accomplishments and prospects. *Transportation Planning and Technology*, 31(3):277–302, 2008.

[11] F. Carrabs, R. Cerulli, and J.-F. Cordeau. An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with LIFO or FIFO loading. *INFOR*, 45(4):223–238, November 2007.

[12] F. Carrabs, J.-F. Cordeau, and G. Laporte. Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. *INFORMS Journal on Computing*, 19: 618–632, 2007.

[13] L. Cassani. Algoritmi euristici per il TSP with rear-loading. Master's thesis, Università degli Studi di Milano, 2004.

[14] A. Ceder, B. Golany, and O. Tal. Creating bus timetables with maximal synchronization. *Transportation Research Part A*, 35:913–928, 2001.

[15] F. Cevallos and F. Zhao. Minimizing transfer times in public transit network with genetic algorithm. *Transportation Research Record*, 1971:74–79, 2006.

[16] P. Chakroborty, K. Deb, and R. K. Sharma. Optimal fleet size distribution and scheduling of transit systems using genetic algorithms. *Transportation Planning and Technology*, 24(3):209–225, 2001.

[17] S. M. Chowdhury and S. I.-J. Chien. Intermodal transit system coordination. *Transportation Planning and Technology*, 25(4):257–287, 2002.

[18] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12: 568–581, 1964.

[19] J.-F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, and J.-S. Sormany. New heuristics for the vehicle routing problem. In A. Langevin and D. Riopel, editors, *Logistics Systems: Design and Optimization*, chapter 9, pages 279–297. Springer, 2005.

[20] J.-F. Cordeau, G. Laporte, J.-Y. Potvin, and M. W. P. Savelsbergh. Transportation on demand. In C. Barnhart and G. Laporte, editors, *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, chapter 7, pages 429–466. Elsevier, Amsterdam, 2007.

[21] J.-F. Cordeau, G. Laporte, and S. Røpke. Recent models and algorithms for one-to-one pickup and delivery problems. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem:*

Latest Advances and New Challenges, chapter 15, pages 327–357. Springer, 2008.

[22] J.-F. Cordeau, M. Dell'Amico, and M. Iori. Branch-and-cut algorithm for the pickup and delivery traveling salesman problem with FIFO loading. Technical report, DISMI, University of Modena and Reggio Emilia, 2009.

[23] J.-F. Cordeau, M. Iori, G. Laporte, and J. J. Salazar-González. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. *Networks*, ?:?–?, forthcoming.

[24] T. G. Crainic and K. H. Kim. Intermodal transportation. In C. Barnhart and G. Laporte, editors, *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, chapter 8, pages 467–537. Elsevier, Amsterdam, 2006.

[25] J. R. Daduna and S. Voß. Practical experiences in schedule synchronization. In I. B. J.R. Daduna and J. Paixão, editors, *Computer-Aided Transit Scheduling: Proceedings of the 6th International Workshop on Computer-aided Scheduling of Public Transport*, pages 39–55. Springer, 1995.

[26] G. Desaulniers and M. Hickman. Public transit. In G. Laporte and C. Barnhart, editors, *Transportation*, Handbooks in Operations Research and Management Science, chapter 3, pages 69–127. Elsevier, 2007.

[27] G. Desaulniers, J. Lavigne, and F. Soumis. Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research*, 111(3):479–494, 1998.

[28] G. Desaulniers, J. Desrosiers, A. Erdmann, M. M. Solomon, and F. Soumis. VRP with pickup and delivery. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 9, pages 225–242. SIAM Monographs on Discrete Mathematics and Applications 9, 2002.

[29] J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 2, pages 35–140. Elsevier, 1995.

[30] K. F. Doerner, G. Fuellerer, R. F. Hartl, M. Gronalt, and M. Iori. Metaheuristics for the vehicle routing problem with loading constraints. *Networks*, 49(4):294–307, 2007.

[31] C. Duhamel, J.-Y. Potvin, and J.-M. Rousseau. A tabu search heuristic for the vehicle routing problem with backhauls and time windows. *Transportation Science*, 31(1):49–59, 1997.

[32] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22(4):425–460, 2000.

[33] G. Erdoğan, J.-F. Cordeau, and G. Laporte. The pickup and delivery traveling salesman problem with first-in-first-out loading. *Computers & Operations Research*, 36:1800–1808, 2009.

[34] A. Felipe, M. T. Ortuño, and G. Tirado. Neighborhood structures to solve the double TSP with multiple stacks using local search. In *Proceedings of FLINS 2008*, 2008.

[35] A. Felipe, M. T. Ortuño, and G. Tirado. The double traveling salesman problem with multiple stacks: a variable neighborhood search approach. *Computers & Operations Research*, 36(11):2983–2993, 2009.

[36] F. Ficarelli. Algoritmi di programmazione matematica per il "TSP with rear loading". Master's thesis, Università degli Studi di Milano, 2005.

[37] C. Fleurent, R. Lessard, and L. Séguin. Transit timetable synchronization: Evaluation and optimization. Technical report, GIRO, 75, rue de Port-Royal Est, Bureau 500, Montreal (Quebec), 2007.

[38] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[39] M. Gendreau. An introduction to tabu search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, chapter 2, pages 37–54. Kluwer Academic Publishers, 2003.

[40] M. Gendreau, A. Hertz, and G. Laporte. The traveling salesman problem with backhauls. *Computers & Operations Research*, 23(5): 501–508, 1996.

[41] M. Gendreau, G. Laporte, and A. Hertz. An approximation algorithm for the traveling salesman problem with backhauls. *Operations Research*, 45(4):639–641, 1997.

[42] M. Gendreau, G. Laporte, and D. Vigo. Heuristics for the traveling salesman problem with pickup and delivery. *Computers & Operations Research*, 26(7):699–714, 1999.

[43] M. Gendreau, G. Laporte, and J.-Y. Potvin. Metaheuristics for the capacitated VRP. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 6. SIAM Monographs on Discrete Mathematics and Applications 9, 2002.

[44] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40(3):342–350, 2006.

[45] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks*, 51 (1):4–18, 2008.

[46] M. Gendreau, J.-Y. Potvin, O. Bräysy, G. Hasle, and A. Løkketangen. Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem*, chapter 7, pages 143–170. Springer, 2008.

[47] G. Ghiani, G. Laporte, and F. Semet. The black and white traveling salesman problem. *Operations Research*, 54(2):366–378, 2006.

[48] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5): 533–549, 1986.

[49] F. W. Glover and G. A. Kochenberger, editors. *Handbook of Metaheuristics*. International Series in Operations Research & Management Science. Springer, January 2003.

[50] A. Goel and V. Gruhn. A general vehicle routing problem. *European Journal of Operational Research*, 191(3):650–660, 2008.

[51] B. Golden and E. Raghavan, S. & Wasil, editors. *The Vehicle Routing Problem — Latest Advances and New Challenges*. Springer, 2008.

[52] I. Gribkovskaia, G. Laporte, and A. Shyshou. The single vehicle routing problem with deliveries and selective pickups. *Computers & Operations Research*, 35(9):2908 – 2924, 2008. Part Special Issue: Bio-inspired Methods in Combinatorial Optimization.

[53] V. Guihaire and J.-K. Hao. Transit network re-timetabling and vehicle scheduling. *Communications in Computer and Information Science*, 14:135–144, 2008. Forthcoming.

[54] V. Guihaire and J.-K. Hao. Transit network design and scheduling: A global review. *Transportation Research Part A*, 42(10):1251–1273, 2008.

[55] A. Hadjar and F. Soumis. Dynamic window reduction for the multiple depot vehicle scheduling problem with time windows. *Computers and Operations Research*, 36(7):2160–2172, 2009.

[56] A. Hadjar, O. Marcotte, and F. Soumis. A branch-and-cut algorihtm for the multiple depot scheduling problem. *Operations Research*, 54(1):130–149, 2006.

[57] P. Hansen and N. Mladenović. Variable neighborhood search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, chapter 6, pages 145–184. Kluwer Academic Publishers, 2003.

[58] G. Hasle. Heuristics for rich VRP models. In *Route 2003, proceedings*, 2003.

[59] D. Henderson, S. H. Jacobson, and A. W. Johnson. The theory and practice of simulated annealing. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, chapter 10, pages 287–319. Kluwer Academic Publishers, 2003.

[60] A. P. R. van den Heuvel, J. M. van den Akker, and M. E. van Koten Niekerk. Integrating timetabling and vehicle scheduling in public bus transportation. Technical Report UU-CS-2008-003, Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, February 2008.

[61] M. Iori, J. J. Salazar-González, and D. Vigo. An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science*, 41(2):253–264, 2007.

[62] L. N. Jansen and M. B. Pedersen. Minimering af skiftetider i køreplaner. Master's thesis, Centre for Traffic & Transport, Technical University of Denmark, March 2002. In Danish.

[63] M. Jünger, G. Reinelt, and G. Grinaldi. The traveling salesman problem. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 4, pages 225–330. Elsevier Science, 1995.

[64] B. Kalantari, A. V. Hill, and S. R. Arora. An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, 22(3):377–386, 1985.

[65] B. Kallehauge, N. Boland, and O. B. G. Madsen. Path inequalities for the vehicle routing problem with time windows. *Networks*, 49 (4):273–293, 2007.

[66] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[67] W.-D. Klemt and W. Stemme. Schedule synchronization for public transit networks. In J. Daduna and A. Wren, editors, *Computer-Aided Transit Scheduling, Proceedings of the Fourth International Workshop on Computer-Aided Scheduling of Public Transport*, pages 327–335. Springer, 1988.

[68] L. Kroon, D. Huisman, and G. Maróti. Railway timetabling from an operations research perspective. Technical Report EI2007-22, Econometric Institute, 2007.

[69] S. P. Ladany and A. Mehrez. Optimal routing of a single vehicle with loading and unloading constraints. *Transportation Planning and Technology*, 8(4):301–306, 1984.

[70] Y. H. Lee, J. W. Jung, and K. M. Lee. Vehicle routing scheduling for cross-docking in the supply chain. *Computers & Industrial Engineering*, 51(2):247–256, 2006.

[71] G. Levitin and R. Abezgaouz. Optimal routing of multiple-load AGV subject to LIFO loading constraints. *Computers & Operations Research*, 30(3):397–410, 2003.

[72] C. Liebchen and R. H. Möhring. A case study in periodic timetabling. *Electronic Notes in Theoretical Computer Science*, 66(6): 18–31, 2002.

[73] C. Liebchen and R. H. Möhring. The modeling power of the periodic event scheduling problem: Railway timetables — and beyond. In F. Geraets, editor, *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 3–40. Springer, 2007.

[74] C. Liebchen and L. Peeters. Some practical aspects of periodic timetabling. In P. Chmoni, R. Leisten, A. Martin, J. Minnemann, and H. Stadtler, editors, *Operations Research 2001*, Heidelberg, 2002. Springer.

[75] A. Löbel. Solving large-scale multiple-depot vehicle scheduling problems. In N. Wilson, editor, *Computer-Aided Transit Scheduling*, pages 193–220. Springer, Berlin, 1999.

[76] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, chapter 11, pages 321–353. Kluwer Acedemic Publishers, 2003.

[77] C. Macharis and Y. M. Bontekoning. Opportunities for OR in intermodal freight transport research: A review. *European Journal of Operational Research*, 153(2):400–416, 2004.

[78] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulations and traveling salesman problems. *Journal of the ACM*, 7:326–329, 1960.

[79] A. Mingozzi, L. Bianco, and S. Ricciardelli. An exact algorithm for combining vehicle trips. In *Computer-aided transit scheduling*, pages 145–172, 1995.

[80] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24 (11):1097–1100, 1997.

[81] F. A. T. Montané and R. D. Galvão. A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. *Computers & Operations Research*, 33(3):595–619, 2006.

[82] G. Mosheiov. Vehicle routing with pick-up and delivery: tour-partitioning heuristics. *Computers & Industrial Engineering*, 34 (3):669–684, 1998.

[83] J. A. Pacheco. Problemas de rutas con carga y descarga en sistemas LIFO: soluciones exactas. *Estudios de Economía Aplicada*, 3:69–86, June 1995.

[84] J. A. Pacheco. Heurístico para los problemas de rutas con carga y descarga en sistemas LIFO. *Qüestiió*, 21:153–175, 1997.

[85] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery models, part i. *Journal für Betriebswirtschaft*, 58: 21–51, 2008.

[86] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery models, part ii. *Journal für Betriebswirtschaft*, 58: 81–117, 2008.

[87] M. B. Pedersen. Minimizing passenger transfer times in public transport timetables. *Orbit*, pages 13–20, 2003. Special ISMP issue of the newsletter of the Danish Operations Research Society.

[88] L. W. P. Peeters. *Cyclic Railway Timetable Optimization*. PhD thesis, Erasmus Research Institute of Management (ERIM), 2003.

[89] A.-S. Pepin, G. Desaulniers, A. Hertz, and D. Huisman. A comparison of five heuristics for the multiple depot vehicle scheduling problem. *Journal of Scheduling*, 12(1):17–30, 2009.

[90] H. L. Petersen. Heuristic solution approaches to the double TSP with multiple stacks. Technical Report 2006-2, Centre for Traffic and Transport, Technical University of Denmark, 2006.

[91] H. L. Petersen and O. B. G. Madsen. The double travelling salesman problem with multiple stacks—formulation and heuristic solution approaches. *European Journal of Operational Research*, 198 (1):139–147, 2009.

[92] H. L. Petersen, A. Larsen, O. B. G. Madsen, and S. Røpke. A data set for the simultaneous vehicle scheduling and passenger service problem. Technical Report 2008-8, DTU Transport, Technical University of Denmark, 2008.

[93] D. Pisinger and S. Røpke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.

[94] J.-Y. Potvin, C. Duhamel, and F. Guertin. Genetic algorithm for vehicle routing with backhauling. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies*, 6(4):345–355, 1996.

[95] R. Ravia and A. Sinha. Approximating k-cuts using network strength as a lagrangean relaxation. *European Journal of Operational Research*, 186:77–90, 2008.

[96] S. Røpke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[97] S. Røpke, J.-F. Cordeau, and G. Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

[98] K. S. Ruland and E. Y. Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & Mathematics with Applications*, 33(12):1–13, 1997.

[99] S. Salhi and G. Nagy. A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *Journal of the Operational Research Society*, 50(10):1034–1042, 1999.

[100] M. W. P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

[101] M. W. P. Savelsbergh and M. Sol. Drive: Dynamic routing of independent vehicles. *Operations Research*, 46(4):474–490, 1998.

[102] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2 (4):550–581, 1989.

[103] J. S. Shang and C. K. Cuff. Multicriteria pickup and delivery problem with transfer opportunity. *Computers & Industrial Engineering*, 30(4):631–645, 1996.

[104] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.-F. Puget, editors, *Principle and Practice of Constraint Programming—CP98*. Springer-Verlag, 1998.

[105] P. Shrivastava and S. Dhingra. Development of coordinated schedules using genetic algorithms. *Journal of Transportation Engineering*, 128(1):89–96, 2002.

[106] B. Suman and P. Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57(10):1143, 2006.

[107] P. Toth and D. Vigo. An exact algorithm for the vehicle routing problem with backhauls. *Transportation Science*, 31(4):372–385, 1997.

[108] P. Toth and D. Vigo. A heuristic algorithm for the symmetric and asymmetric vehicle routing problems with backhauls. *European Journal of Operational Research*, 113(3):528–543, 1999.

[109] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. Number 9 in SIAM Monographs on Discrete Mathematics and Applications. SIAM, 2002.

[110] P. Toth and D. Vigo. VRP with backhauls. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 8, pages 195–224. SIAM Monographs on Discrete Mathematics and Applications 9, 2002.

[111] F. Tricoire, K. F. Doerner, R. F. Hartl, and M. Iori. Heuristic and exact algorithms for the multi-pile vehicle routing problem, 2008.

[112] M. Wen, J. Larsen, J. Clausen, J.-F. Cordeau, and G. Laporte. Vehicle routing with cross-docking. *Journal of the Operational Research Society*, ?:?–?, forthcoming.

[113] R. Wong, T. Yuen, K. Fung, and J. Leung. Optimizing timetable synchronization for rail mass transit. *Transportation Science*, 42 (1):57–69, 2008.

[114] H. Xu, Z.-L. Chen, S. Rajagopal, and S. Arunapuram. Solving a practical pickup and delivery problem. *Transportation Science*, 37 (3):347–364, 2003.

[115] Y. Zhong and M. H. Cole. A vehicle routing problem with backhauls and time windows: a guided local search solution. *Transportation Research Part E: Logistics and Transportation Review*, 41(2):131–144, 2005.